# Expeiment 5

Aim

Study of Network Simulator (NS)

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.

Basic Architecture

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. The C++ and the OTcl are linked together using TclCL
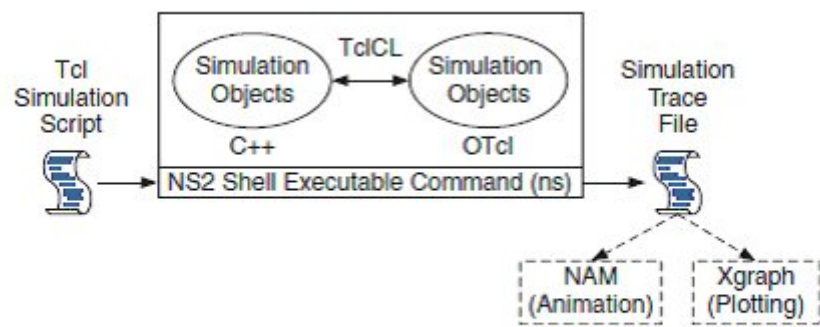
Basic architecture of NS.

TCL

Tcl (pronounced "tickle" or as an initialism) is a high-level, general-purpose, interpreted, dynamic programming language. It was designed with the goal of being very simple but powerful. Tcl casts everything into the mold of a command, even programming constructs like variable assignment and procedure definition.

Basic Syntax

Tcl commands are composed of words separated by white space. The first word is the name of the command and the remaining words are arguments to the command. The following example shows how to use the puts command to print a string:

```
puts "Hello World"
```

Comments

Tcl supports two types of comments: single line comments and multi-line comments. Single line comments start with a hash sign (#) and end with a newline character. Multi-line comments start with the string /*and end with the string*/. The following example shows how to use comments:

```
# This is a single line comment
puts "Hello World" # This is a single line comment
puts "Hello World" /* This is a multi-line comment */ puts "Hello World"
```

Variables

Variables are used to store values. The value of a variable is substituted by its name preceded by a dollar sign. The value of a variable is set using the set command. The set command takes two arguments: the name of the variable and the value to be stored in the variable. The value of a variable can be retrieved using the set command with only one argument: the name of the variable. The following example shows how to use variables:

```
set a 10
set b 20
```

Control Structures

Tcl has the usual control structures such as if, while, for, and switch. The if command takes a condition and a body of code to execute if the condition is true. The condition is an expression that evaluates to a boolean value. The body of code is a Tcl script enclosed in braces. The following example shows how to use the if

command:

```
if {$a < $b} {
    puts "$a is less than $b"
} else {
    puts "$a is greater than or equal to $b"
}
```

Initialization and Termination of TCL Script in NS-2

To initialize an NS simulation, the first line in the TCL script should declare a new variable using the set command as follows:

```
set ns [new Simulator]
```

The variable ns represents an instance of the Simulator class, which is created using the reserved word new. To create output files for simulation trace data and visualization, use the open command as follows:

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

These commands create a data trace file called out.tr and a visualization trace file called out.nam. The files are not called explicitly by name within the TCL script, but by pointers declared above as tracefile1 and namfile, respectively. Note that these pointers begin with the # symbol. The trace-all method is used to record all simulation traces in the out.tr file, and the namtrace-all method is used to record the visualization trace in the out.nam file. To terminate the program, use the finish procedure as follows:

```
# Define a 'finish' procedure
proc finish {} {
  global ns tracefile1 namfile
  $ns flush-trace
  close $tracefile1
  close $namfile
  exec nam out.nam &
  exit 0
}
```

The proc command declares a procedure called finish, which takes no arguments. The global command is used to indicate that variables declared outside the procedure will be used. The flush-trace method dumps the traces on the respective files, and the close command closes the trace files defined earlier. The exec command executes the nam program for visualization, and the exit command ends the application and returns the number 0 as a status to the system. At the end of the NS program, call the finish procedure and specify the time at which the termination should occur, as follows:

```
$ns at 125.0 "finish"
```

This command connects nodes $n0 and $n2 using a bi-directional link with a propagation delay of 10ms and a capacity of 10Mb per second for each direction. To define a directional link instead of a bi-directional one, replace duplex-link with simplex-link.

To define an output queue of a node, include it as part of each link whose input is that node. The following command sets the buffer capacity of the queue related to a link:

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

To define routing (sources, destinations), agents (protocols), and applications that use them, follow the examples below.

For FTP over TCP:

```
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
```

```
$tcp set fid_1
$tcp set packetSize_ 552
```

For CBR over UDP:

```
set udp [new Agent/UDP] # Create a UDP agent
$ns attach-agent $n1 $udp # Define the source node of the UDP connection
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null # Define the behavior of the destination node of UDP
$udp set fid_2 # Set the flow identification of the UDP connection to '2'
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp # Create a CBR application using the UDP agent
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false # Define the destination node of the CBR application
$ns attach-agent $n4 $sink # Schedule the beginning and end of the CBR application
$ns at 0.1 "$cbr start"
$ns at 124.5 "$cbr stop"
```

To create a link between two nodes, use the `duplex-link` method of the simulator as follows:

```
$ns duplex-link <node1> <node2> <propagation delay> <queue>
```

The `<queue>` argument is optional. If it is not specified, the link is assumed to have infinite capacity. To schedule events, use the `at` method of the simulator as follows:

```
$ns at <time> <event>
```

Result

We have studied the Network Simulator 2 (NS2) in detail.

Experiment 6 (a)

Aim

To simulate and study the Distance Vector routing algorithm using simulation using NS2.

Theory

Introduction

Routing is the process of selecting the best paths in a network. In electronic data networks using packet switching technology, routing directs packet forwarding through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Most routing algorithms use only one network path at a time. However, multipath routing techniques enable the use of multiple alternative paths.

Algorithm

The Distance Vector routing protocol works as follows:

1. Each router maintains a table that lists the cost of reaching each destination network in the internetwork.
2. Each router periodically shares its routing table with its neighboring routers.
3. Each router updates its own routing table based on the information it receives from its neighbors.
4. The router selects the path with the lowest cost to the destination network as the best path.
5. The router forwards packets to the next hop router in the best path.

Program

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tr [open out.tr w]
$ns trace-all $tr

proc finish {} {
    global nf ns tr
    $ns flush-trace
    close $tr
    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 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail

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

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

set udp [new Agent/UDP]
$ns attach-agent $n2 $udp

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp

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

$ns connect $tcp $sink
```

```
$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto DV

$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"
$ns at 5.0 "finish"

$ns run
```

Result

The Distance Vector Routing Algorithm was successfully simulated and studied using NS2.

Experiment 6 (b)

Aim

To simulate and study the Link State Routing algorithm using NS2.

Link State Routing Protocol

In Link State Routing, each router shares its knowledge of its neighborhood with every other router in the internetwork. The protocol works as follows:

1. Knowledge about Neighborhood: Instead of sending its entire routing table, a router sends information about its neighborhood only.
2. To all Routers: Each router sends this information to every other router on the internet work, not just to its neighbor. It does so by a process called flooding.
3. Information sharing when there is a change: Each router sends out information about the neighbors when there is a change.

Algorithm

The following algorithm is used to simulate the Link State Routing protocol:

1. Create a simulator object.
2. Define different colors for different data flows.
3. Open a nam trace file and define a finish procedure. Then close the trace file and execute nam on trace file.
4. Create n number of nodes using a for loop.
5. Create duplex links between the nodes.
6. Setup UDP Connection between n(0) and n(5).
7. Setup another UDP connection between n(1) and n(5).
8. Apply CBR Traffic over both UDP connections.
9. Choose Link State Routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

Program

```tcl
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tr [open out.tr w]
$ns trace-all $tr

proc finish {} {
    global nf ns tr
    $ns flush-trace
    close $tr
    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 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail

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

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

set udp [new Agent/UDP]
$ns attach-agent $n2 $udp

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
```

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

$ns connect $tcp $sink
$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto LS

$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"
$ns at 5.0 "finish"

$ns run
```

Result

The Link State Routing Algorithm was successfully simulated and studied using NS2.

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


$ns connect $tcp $sink
$ns connect $udp $null


$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3


$ns rtproto LS


$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"
```

Experiment 7

Aim

To implement Stop and Wait Protocol and Sliding Window Protocol.

Code

Sender

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

class StopWaitSender {
    public static void main(String args[]) throws Exception {
        StopWaitSender sws = new StopWaitSender();
        sws.run();
    }
    public void run() throws Exception {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter no of frames to be sent:");
        int n = sc.nextInt();
        Socket myskt = new Socket("localhost", 9999);
        PrintStream myps = new PrintStream(myskt.getOutputStream());
        for (int i = 0; i <= n;) {
            if (i == n) {
                myps.println("exit");
                break;
            }
            System.out.println("Frame no " + i + " is sent");
            myps.println(i);
            BufferedReader bf = new BufferedReader(new InputStreamReader(myskt.getInputStream()));
            String ack = bf.readLine();
            if (ack != null) {
                System.out.println("Acknowledgement was Received from receiver");
                i++;
                Thread.sleep(4000);
            } else {  myps.println(i);
            }
        }
    }
}
```

Receiver

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

class StopWaitReceiver {
    public static void main(String args[]) throws Exception {
        StopWaitReceiver swr = new StopWaitReceiver();
        swr.run();
    }
    public void run() throws Exception {
        String temp = "any message";
        String str = "exit";
        ServerSocket myss = new ServerSocket(9999);
        Socket ss_accept = myss.accept();
        BufferedReader ss_bf = new BufferedReader(new InputStreamReader(ss_accept.getInputStream()));
        PrintStream myps = new PrintStream(ss_accept.getOutputStream());

        while (temp.compareTo(str) != 0) {
            Thread.sleep(1000);
            temp = ss_bf.readLine();
            if (temp.compareTo(str) == 0) {
                break;
            }
            System.out.println("Frame " + temp + " was received");
            Thread.sleep(500);
            myps.println("Received");
        }
```

```
            System.out.println("ALL FRAMES WERE RECEIVED SUCCESSFULLY");
        }
    }
```

Output

Sender

```
Enter no of frames to be sent:
4
Frame no 0 is sent
Acknowledgement was Received from receiver
Frame no 1 is sent
Acknowledgement was Received from receiver
Frame no 2 is sent
Acknowledgement was Received from receiver
Frame no 3 is sent
Acknowledgement was Received from receiver
```

Receiver

```
Frame 0 was received
Frame 1 was received
Frame 2 was received
Frame 3 was received
ALL FRAMES WERE RECEIVED SUCCESSFULLY
```

Sliding Window Protocol

Sliding Window Protocol is a flow control protocol in which a sender sends multiple packets to the receiver before receiving the acknowledgment of the first few packets.

Code

Bit Client

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

class BitClient {
    public static void main(String[] args) {
        try {
            BufferedInputStream in;
            ServerSocket serverSocket = new ServerSocket(500);
            System.out.println("Waiting for connection");
            Socket client = serverSocket.accept();
            System.out.println("Received request for sending frames");
            in = new BufferedInputStream(client.getInputStream());
            DataOutputStream out = new DataOutputStream(client.getOutputStream());
            int p = in.read();
            System.out.println("Sending .... ");

            for (int i = 1; i <= p; ++i) {
                System.out.println("Sending frame no " + i);
                out.write(i);
                out.flush();
                System.out.println("Waiting for acknowledge");
                Thread.sleep(5000);
                int a = in.read();
                System.out.println("Received acknowledge for frame no: " + i + " as " + a);
            }

            out.flush();
            in.close();
            out.close();
            client.close();
            serverSocket.close();
            System.out.println("Quiting");
        } catch (IOException e) {
```

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

Bit Server

```java
import java.lang.System;
import java.net.*;
import java.io.*;
import java.math.*;

class BitServer {
    public static void main(String a[]) {
        try {
            InetAddress addr = InetAddress.getByName("localhost");
            System.out.println(addr);
            Socket connection = new Socket(addr, 500);
            DataOutputStream out = new DataOutputStream(connection.getOutputStream());
            BufferedInputStream in = new BufferedInputStream(connection.getInputStream());
            BufferedInputStream inn = new BufferedInputStream(connection.getInputStream());
            BufferedReader ki = new BufferedReader(new InputStreamReader(System.in));
            int flag = 0;
            System.out.println("Connect");
            System.out.println("Enter the no of frames to be requested to server:");
            int c = Integer.parseInt(ki.readLine());
            out.write(c);
            out.flush();
            int i, jj = 0;

            while (jj < c) {
                i = in.read();
                System.out.println("Received frame no " + i);
                System.out.println("Sending acknowledgement for frame no " + i);
                out.write(i);
                out.flush();
                jj++;
            }

            out.flush();
            in.close();
            inn.close();
            out.close();

            System.out.println("Output:");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output

Bit Client

```
Waiting for connection
Received request for sending frames
Sending ....
Sending frame no 1
Waiting for acknowledge
Received acknowledge for frame no: 1 as 1
Sending frame no 2
Waiting for acknowledge
Received acknowledge for frame no: 2 as 2
Sending frame no 3
Waiting for acknowledge
Received acknowledge for frame no: 3 as 3
Sending frame no 4
Waiting for acknowledge
Received acknowledge for frame no: 4 as 4
Quiting
```

Bit Server

```
localhost/127.0.0.1
Connect
Enter the no of frames to be requested to server:
4
Received frame no 1
Sending acknowledgement for frame no 1
Received frame no 2
Sending acknowledgement for frame no 2
Received frame no 3
Sending acknowledgement for frame no 3
Received frame no 4
Sending acknowledgement for frame no 4
Output:
```

Conclusion

Thus, we have studied the various types of flow control protocols and implemented them using Java.

EXPERIMENT 8

Aim

The aim of this experiment is to study Socket Programming and Client-Server model.

Theory

Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connection-less. The client in socket programming must know two pieces of information: IP address of the server and port number. Here, we are going to make one-way client and server communication. In this application, the client sends a message to the server, the server reads the message, and prints it.

SOCKET API
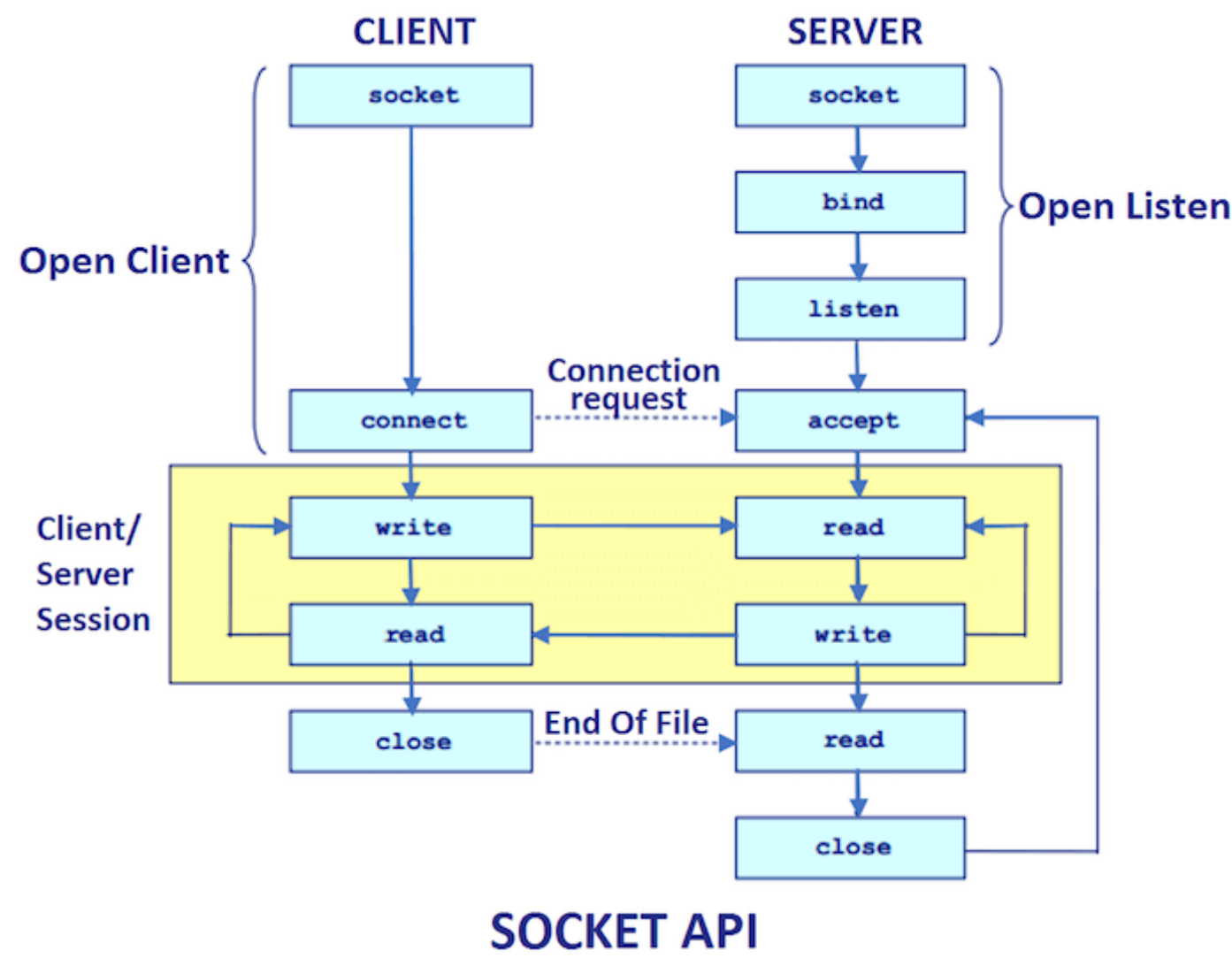
Socket Class

A socket is simply an endpoint for communications between machines. The Socket class can be used to create a socket. Important methods include:

- `public InputStream getInputStream()`: returns the InputStream attached with this socket.
- `public OutputStream getOutputStream()`: returns the OutputStream attached with this socket.
- `public synchronized void close()`: closes this socket.

ServerSocket Class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with clients. Important methods include:

- `public Socket accept()`: returns the socket and establish a connection between server and client.
- `public synchronized void close()`: closes the server socket.

Code

Server.java

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

public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(6666);
            Socket s = ss.accept(); // establishes connection
            DataInputStream dis = new DataInputStream(s.getInputStream());
            String str = (String) dis.readUTF();
            System.out.println("message= " + str);
            ss.close();
```

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

Client.java

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

public class Client {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("localhost", 6666);
            DataOutputStream dout = new DataOutputStream(s.getOutputStream());
            dout.writeUTF("Hello Server");
            dout.flush();
            dout.close();
            s.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Output

Server

```
PROBLEMS  43    PORTS  4    TERMINAL    OUTPUT    COMMENTS    DEBUG CONSOLE

● @chirag127 →/workspaces/6th-Sem-CN-Computer-Networks/practicle/8 (main) $ cd "/workspaces/6th-Sem-CN-Computer-Networks/practicle/8/" && javac
  Server.java && java Server         I
  message= Hello Server
○ @chirag127 →/workspaces/6th-Sem-CN-Computer-Networks/practicle/8 (main) $ []
```

Client

```
PROBLEMS  23    PORTS  4    TERMINAL    OUTPUT    COMMENTS    DEBUG CONSOLE

● @chirag127 →/workspaces/6th-Sem-CN-Computer-Networks (main) $ cd "/workspaces/6th-Sem-CN-Computer-Networks/practicle/8/" && javac Client.java
  && java Client
○ @chirag127 →/workspaces/6th-Sem-CN-Computer-Networks/practicle/8 (main) $ []
```

Conclusion

Thus, we have studied Socket Programming and Client-Server model in Java. Socket programming provides a way to communicate between different applications running on different JRE. It is important to establish a connection between the client and server, and properly handle the input and output streams.

EXPERIMENT 9

Aim

The aim of this experiment is to simulate ARP / RARP protocols using Java programming language.

Theory

ARP (Address Resolution Protocol) is used to translate the IP address of a host to its corresponding MAC (Media Access Control) address. RARP (Reverse Address Resolution Protocol) is used to translate the MAC address of a host to its corresponding IP address. In this experiment, we will write a Java program that simulates ARP and RARP protocols. This program will automatically create a file with the IP address of machines, their MAC address, and type.

Code

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

public class arp_rarp {
    private static final String Command = "arp -a";

    public static void getARPTable(String cmd) throws Exception {
        File fp = new File("ARPTable.txt");
        FileWriter fw = new FileWriter(fp);
        BufferedWriter bw = new BufferedWriter(fw);
        Process P = Runtime.getRuntime().exec(cmd);
        Scanner S = new Scanner(P.getInputStream()).useDelimiter("\\A");
        while (S.hasNext())
            bw.write(S.next());
        bw.close();
        fw.close();
    }

    public static void findMAC(String ip) throws Exception {
        File fp = new File("ARPTable.txt");
        FileReader fr = new FileReader(fp);
        BufferedReader br = new BufferedReader(fr);
        String line;
        while ((line = br.readLine()) != null) {
            if (line.contains(ip)) {
                System.out.println("Internet Address Physical Address Type");
                System.out.println(line);
                break;
            }
        }
        if ((line == null))
            System.out.println("Not found");
        fr.close();
        br.close();
    }

    public static void findIP(String mac) throws Exception {
        File fp = new File("ARPTable.txt");
        FileReader fr = new FileReader(fp);
        BufferedReader br = new BufferedReader(fr);
        String line;
        while ((line = br.readLine()) != null) {
            if (line.contains(mac)) {
                System.out.println("Internet Address Physical Address Type");
                System.out.println(line);
                break;
            }
        }
        if ((line == null))
            System.out.println("Not found");
        fr.close();
        br.close();
    }

    public static void main(String as[]) throws Exception {
        getARPTable(Command);
        Scanner S = new Scanner(System.in);
        System.out.println("ARP Protocol.");
        System.out.print("Enter IP Address: ");
        String IP = S.nextLine();
```

```
            findMAC(IP);
            System.out.println("RARP Protocol.");
            System.out.print("Enter MAC Address: ");
            String MAC = S.nextLine();
            findIP(MAC);
        }
    }
```

ARPTable.txt

```
Interface: 192.168.1.10 --- 0x2
  Internet Address      Physical Address      Type
  192.168.1.1           00-0e-2e-5c-7a-0a     dynamic
  192.168.1.2           00-15-c5-4b-8f-2f     dynamic
  192.168.1.5           00-21-cc-f8-5e-2f     dynamic
  192.168.1.7           00-1d-d9-b1-7e-50     dynamic
```

Output

```
cd "/workspaces/6th-Sem-CN-Computer-Networks/practicle/9/" && javac arp_rarp.java && java arp_rarp
@chirag127 ➜/workspaces/6th-Sem-CN-Computer-Networks (main) $ cd "/workspaces/6th-Sem-CN-Computer-Networks/practicle/9/" && javac arp_rarp.java && java arp_rarp
ARP Protocol.
Enter IP Address: 192.168.1.1
Internet Address Physical Address Type
Interface: 192.168.1.10 --- 0x2
RARP Protocol.
Enter MAC Address: 00-21-cc-f8-5e-2f
Internet Address Physical Address Type
  192.168.1.5           00-21-cc-f8-5e-2f     dynamic
@chirag127 ➜/workspaces/6th-Sem-CN-Computer-Networks/practicle/9 (main) $ []
```

Conclusion

In this experiment, we have learned about ARP and RARP protocols and how they can be simulated using Java programming language. The program automatically creates a file with the IP address of machines, their MAC address, and type. The program can also be extended to read an IP Address / Mac Address and a message and send a packet to the specified machine using TCP / IP or Datagram sockets.

EXPERIMENT 10 (a)

Aim

The aim of this experiment is to write a Java program that simulates the PING command.

Algorithm

1. Start the program.
2. Import necessary packages in Java.
3. Create a Process object to implement the PING command.
4. Declare a BufferedReader stream class object.
5. Get the details of the server.
6. Print the results.
7. Stop the program.

Code

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

class pingserver {
    public static void main(String args[]) {
        try {
            String str;
            System.out.print("Enter the IP Address to be Ping: ");
            BufferedReader buf1 = new BufferedReader(new InputStreamReader(System.in));
            String ip = buf1.readLine();
            Runtime H = Runtime.getRuntime();
            Process p = H.exec("ping " + ip);
            InputStream in = p.getInputStream();
            BufferedReader buf2 = new BufferedReader(new InputStreamReader(in));
            while ((str = buf2.readLine()) != null) {
                System.out.println(" " + str);
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output

```
Enter the IP Address to be Ping: 192.168.0.1
Pinging 192.168.0.1: with bytes of data =32
Reply from 192.168.0.11: bytes=32 time<1ms TTL=128
Reply from 192.168.0.11: bytes=32 time<1ms TTL=128
Reply from 192.168.0.11: bytes=32 time<1ms TTL=128
Reply from 192.168.0.11: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.0.1:
    Packets: sent=4, received=4, lost=0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 4ms, Average = 2ms
```

Conclusion

In this experiment, we have learned how to simulate the PING command using Java programming language.

EXPERIMENT 10 (b)

Aim

The aim of this experiment is to write a Java program that simulates the TRACERT command.

Theory

The TRACERT command is used to trace the route to a destination IP address or domain name.

Algorithm

1. Start the program.
2. Import necessary packages in Java.
3. Create a method to run system commands.
4. Create a main method to execute the TRACERT command.
5. Prompt the user to enter a domain name or IP address to trace the route to.
6. Execute the TRACERT command using the runSystemCommand method.
7. Print the results.
8. Stop the program.

Code

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class traceroutecmd {
    public static void runSystemCommand(String command) {
        try {
            Process p = Runtime.getRuntime().exec(command);
            BufferedReader inputStream = new BufferedReader(new InputStreamReader(p.getInputStream()));
            String s = "";
            while ((s = inputStream.readLine()) != null)
                System.out.println(s);
        } catch (Exception e) {
        }
    }

    public static void main(String[] args) {
        String ip = "www.drranurekha.com";
        System.out.println("Tracing route to " + ip + " over a maximum of 30 hops:");
        runSystemCommand("tracert " + ip);
    }
}
```

Output

```
Tracing route to drranurekha.com [160.153.137.167] over a maximum of 30 hops:
  1    <1 ms     <1 ms     <1 ms  10.0.15.1
  2    <1 ms     <1 ms     <1 ms  10.0.0.15
  3     1 ms      1 ms      1 ms  210.212.247.209
  4     2 ms      1 ms      1 ms  172.24.75.102
  5    *         *         21 ms  218.248.235.217
  6    *         *         12 ms  218.248.235.218
  7    21 ms     21 ms     21 ms  121.244.37.253.static.chennai.vsnl.net.in [121.244.37.253]
  8    *         *         *      Request timed out.
  9    49 ms     49 ms     49 ms  172.25.81.134
 10    50 ms     50 ms     70 ms  ix-ae-0-4.tcore1.mlv-mumbai.as6453.net [180.87.38.5]
 11   165 ms    165 ms    165 ms  if-ae-9-5.tcore1.wyn-marseille.as6453.net [80.231.217.17]
 12   172 ms    171 ms    171 ms  if-ae-8-1600.tcore1.pye-paris.as6453.net [80.231.217.6]
 13   171 ms    171 ms    171 ms  if-ae-15-2.tcore1.av2-amsterdam.as6453.net [195.219.194.145]
 14   175 ms    175 ms    175 ms  195.219.194.2
 15   171 ms    170 ms    170 ms  po72.bbsa0201-01.bbn.mgmt.ams1.gdg [188.121.33.74]
 16   170 ms    169 ms    169 ms  10.241.131.203
 17   175 ms    175 ms    175 ms  10.253.1.1
 18   166 ms    166 ms    166 ms  10.253.130.9
 19   173 ms    173 ms    173 ms  10.253.130.3
 20   169 ms    169 ms    169 ms  10.253.130.5
 21   169 ms    169 ms    169 ms  ip-160-153-137-167.ip.secureserver.net [160.153.137.167]
```

```
Trace complete.
```

Conclusion

In this experiment, we have learned how to simulate the TRACERT command using Java programming language.

Conclusion

EXPERIMENT 11

Aim

The aim of this experiment is to create a socket for HTTP for web page upload and download.

Theory

Create a socket for HTTP for web page upload and download.

Algorithm

1. Start the program.
2. Get the frame size from the user.
3. Create the frame based on the user request.
4. Send frames to the server from the client side.
5. If the frames reach the server, it will send an ACK signal to the client; otherwise, it will send a NACK signal to the client.
6. Stop the program.

Code

Client

```java
import javax.swing.*;
import java.net.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class Client {
    public static void main(String args[]) throws Exception {
        Socket soc;
        BufferedImage img = null;
        soc = new Socket("localhost", 4000);
        System.out.println("Client is running.");
        try {
            System.out.println("Reading image from disk.");
            img = ImageIO.read(new File("digital_image_processing.jpg"));
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ImageIO.write(img, "jpg", baos);
            baos.flush();
            byte[] bytes = baos.toByteArray();
            baos.close();
            System.out.println("Sending image to server.");
            OutputStream out = soc.getOutputStream();
            DataOutputStream dos = new DataOutputStream(out);
            dos.writeInt(bytes.length);
            dos.write(bytes, 0, bytes.length);
            System.out.println("Image sent to server.");
            dos.close();
            out.close();
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            soc.close();
        }
        soc.close();
    }
}
```

Server

```java
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;
```

```java
class Server {
    public static void main(String args[]) throws Exception {
        ServerSocket server = null;
        Socket socket;
        server = new ServerSocket(4000);
        System.out.println("Server waiting for image.");
        socket = server.accept();
        System.out.println("Client connected.");
        InputStream in = socket.getInputStream();
        DataInputStream dis = new DataInputStream(in);
        int len = dis.readInt();
        System.out.println("Image size: " + len / 1024 + "KB");
        byte[] data = new byte[len];
        dis.readFully(data);
        dis.close();
        in.close();
        InputStream ian = new ByteArrayInputStream(data);
        BufferedImage bImage = ImageIO.read(ian);
        JFrame f = new JFrame("Server");
        ImageIcon icon = new ImageIcon(bImage);
        JLabel l = new JLabel();
        l.setIcon(icon);
        f.add(l);
        f.pack();
        f.setVisible(true);
    }
}
```

Conclusion

In this experiment, we have learned how to create a socket for HTTP for web page upload and download using Java programming language. The program prompts the user to enter a frame size and creates a frame based on the user request. The client then sends the frames to the server using a socket, and the server receives the frames and displays the image in a JFrame. The program can be useful for transmitting images over a network.

EXPERIMENT 12

Aim

The aim of this experiment is to write a program to implement RPC (Remote Procedure Call) in Java.

Introduction

Remote Procedure Call (RPC) is an inter-process communication technique used in client-server applications. It allows a client to call a procedure or a function on a remote server as if it were a local function call. The sequence of events in an RPC involves the client making a request to the server, the server processing the request, and sending back a response to the client.

Code

HelloWorld.java

```java
package rpc_helloworld;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld {
    @WebMethod
    String getHelloWorld(String name);
}
```

HelloWorldImpl.java

```java
package rpc_helloworld;

import javax.jws.WebService;

@WebService(endpointInterface = "rpc_helloworld.HelloWorld")
public class HelloWorldImpl implements HelloWorld {
    @Override
    public String getHelloWorld(String name) {
        return name;
    }
}
```

Publisher.java

```java
package rpc_helloworld;

import javax.xml.ws.Endpoint;

public class Publisher {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:7779/ws/hello", new HelloWorldImpl());
    }
}
```

rpc_helloworld.java

```java
package rpc_helloworld;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
```

```java
public class RPC_HelloWorld {
    public static void main(String[] args) {
        try {
            // Refer to WSDL document
            URL url = new URL("http://localhost:7779/ws/hello?wsdl");

            // Refer to service
            QName qname = new QName("http://rpc_helloworld/", "HelloWorldImplService");
            Service service = Service.create(url, qname);

            HelloWorld hello = service.getPort(HelloWorld.class);
            System.out.println(hello.getHelloWorld("Hello World!"));
        } catch (MalformedURLException ex) {
            System.out.println("WSDL document URL error: " + ex);
        }
    }
}
```

Procedure

1. Create a Java interface named HelloWorld with a method named getHelloWorld.
2. Annotate the interface with @WebService and @SOAPBinding(style = Style.RPC).
3. Create a Java class named HelloWorldImpl that implements the HelloWorld interface.
4. Annotate the HelloWorldImpl class with @WebService(endpointInterface = "rpc_helloworld.HelloWorld").
5. Implement the getHelloWorld method in the HelloWorldImpl class.
6. Create a Java class named Publisher that publishes the HelloWorldImpl web service using Endpoint.publish.
7. Create a Java class named RPC_HelloWorld that invokes the getHelloWorld method using Service.create and service.getPort.
8. Run the Publisher class to publish the web service on http://localhost:7779/ws/hello.
9. Open a web browser and enter the URL http://localhost:7779/ws/hello?wsdl to view the web service WSDL document.
10. Copy the targetNamespace from the WSDL document and paste it into the QName object in the RPC_HelloWorld class.
11. Run the RPC_HelloWorld class to invoke the getHelloWorld method on the remote server.

Conclusion

In this experiment, we have learned how to implement Remote Procedure Call (RPC) in Java. We created a web service using JAX-WS and published it using Endpoint.publish. We then invoked the remote procedure using Service.create and service.getPort. The program can be useful for implementing client-server applications that require remote procedure calls.

EXPERIMENT 13

Aim

The aim of this experiment is to implement subnetting in Java.

Code

```java
import java.util.Scanner;

class Subnet {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the IP address: ");
        String ip = sc.nextLine();
        String split_ip[] = ip.split("\\.");
        String split_bip[] = new String[4];
        String bip = "";
        for (int i = 0; i < 4; i++) {
            split_bip[i] = appendZeros(Integer.toBinaryString(Integer.parseInt(split_ip[i])));
            bip += split_bip[i];
        }
        System.out.println("IP in binary is " + bip);
        System.out.print("Enter the number of addresses: ");
        int n = sc.nextInt();

        // Calculation of mask
        int bits = (int) Math.ceil(Math.log(n) / Math.log(2));
        System.out.println("Number of bits required for address = " + bits);
        int mask = 32 - bits;
        System.out.println("The subnet mask is = " + mask);

        // Calculation of first address and last address
        int fbip[] = new int[32];
        for (int i = 0; i < 32; i++) {
            fbip[i] = (int) bip.charAt(i) - 48; // Convert character 0,1 to integer 0,1
        }
        for (int i = 31; i > 31 - bits; i--) { // Get first address by ANDing last n bits with 0
            fbip[i] &= 0;
        }
        String fip[] = { "", "", "", "" };
        for (int i = 0; i < 32; i++) {
            fip[i / 8] = new String(fip[i / 8] + fbip[i]);
        }
        System.out.print("First address is = ");
        for (int i = 0; i < 4; i++) {
            System.out.print(Integer.parseInt(fip[i], 2));
            if (i != 3)
                System.out.print(".");
        }
        System.out.println();

        int lbip[] = new int[32];
        for (int i = 0; i < 32; i++) {
            lbip[i] = (int) bip.charAt(i) - 48; // Convert character 0,1 to integer 0,1
        }
        for (int i = 31; i > 31 - bits; i--) { // Get last address by ORing last n bits with 1
            lbip[i] |= 1;
        }
        String lip[] = { "", "", "", "" };
        for (int i = 0; i < 32; i++) {
            lip[i / 8] = new String(lip[i / 8] + lbip[i]);
        }
        System.out.print("Last address is = ");
        for (int i = 0; i < 4; i++) {
            System.out.print(Integer.parseInt(lip[i], 2));
            if (i != 3)
                System.out.print(".");
        }
        System.out.println();
    }

    static String appendZeros(String s) {
        String temp = new String("00000000");
```
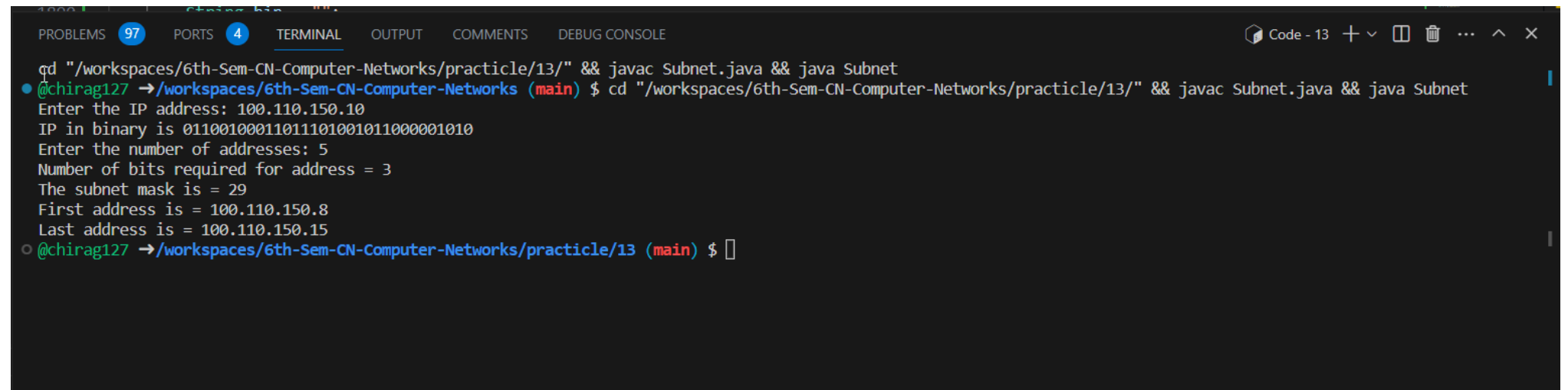
```
            return temp.substring(s.length()) + s;
        }
    }
```

Output

Conclusion

In this experiment, we implemented subnetting in Java.

EXPERIMENT 14

Aim

The aim of this experiment is to implement TCP/IP echo algorithm using Java sockets.

Server

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

Client

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

Code

TCP Echo Server - tcpechoserver.java

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

public class tcpechoserver {
    public static void main(String[] arg) throws IOException {
        ServerSocket sock = null;
        BufferedReader fromClient = null;
        OutputStreamWriter toClient = null;
        Socket client = null;
        try {
            sock = new ServerSocket(4000);
            System.out.println("Server Ready");
            client = sock.accept();
            System.out.println("Client Connected");
            fromClient = new BufferedReader(new InputStreamReader(client.getInputStream()));
            toClient = new OutputStreamWriter(client.getOutputStream());
            String line;
            while (true) {
                line = fromClient.readLine();
                if ((line == null) || line.equals("bye"))
                    break;
                System.out.println("Client [ " + line + " ]");
                toClient.write("Server [ " + line + " ]\n");
                toClient.flush();
            }
            fromClient.close();
            toClient.close();
            client.close();
            sock.close();
            System.out.println("Client Disconnected");
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

TCP Echo Client - tcpechoclient.java

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

public class tcpechoclient {
    public static void main(String[] args) throws IOException {
        BufferedReader fromServer = null, fromUser = null;
        PrintWriter toServer = null;
        Socket sock = null;
        try {
            if (args.length == 0)
                sock = new Socket(InetAddress.getLocalHost(), 4000);
            else
                sock = new Socket(InetAddress.getByName(args[0]), 4000);
            fromServer = new BufferedReader(new InputStreamReader(sock.getInputStream()));
            fromUser = new BufferedReader(new InputStreamReader(System.in));
            toServer = new PrintWriter(sock.getOutputStream(), true);
            String Usrmsg, Srvmsg;
            System.out.println("Type \"bye\" to quit");
            while (true) {
                System.out.print("Enter msg to server : ");
                Usrmsg = fromUser.readLine();
                if (Usrmsg == null || Usrmsg.equals("bye")) {
                    toServer.println("bye");
                    break;
                } else
                    toServer.println(Usrmsg);
                Srvmsg = fromServer.readLine();
                System.out.println(Srvmsg);
            }
            fromUser.close();
            fromServer.close();
            toServer.close();
            sock.close();
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

Output

Server

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

Client

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

Conclusion

The TCP/IP echo algorithm was successfully implemented using Java sockets. The server listens for incoming connections and echoes back whatever data it receives from the client until the client sends the "bye" or "null" message. The client sends data to the server and displays the echoed data until the user types "bye" or "null".

EXPERIMENT 15

Aim

The aim of this experiment is to implement DNS (Domain Name System) in Java.

Code

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

public class DNS {
    public static void main(String[] args) {
        int n;
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        do {
            System.out.println("\n Menu: \n 1. DNS\n 2. Reverse DNS\n 3. Exit \n");
            System.out.println("\n Enter your choice");
            n = Integer.parseInt(System.console().readLine());
            if (n == 1) {
                try {
                    System.out.println("\n Enter Host Name ");
                    String hname = in.readLine();
                    InetAddress address;
                    address = InetAddress.getByName(hname);
                    System.out.println("Host Name: " + address.getHostName());
                    System.out.println("IP: " + address.getHostAddress());
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                }
            }
            if (n == 2) {
                try {
                    System.out.println("\n Enter IP address");
                    String ipstr = in.readLine();
                    InetAddress ia = InetAddress.getByName(ipstr);
                    System.out.println("IP: " + ipstr);
                    System.out.println("Host Name: " + ia.getHostName());
                } catch (IOException ioe) {
                    ioe.printStackTrace();
                }
            }
        } while (!(n == 3));
    }
}
```

Explanation

The program starts by taking input from the user in the form of a menu. The user is given the option to choose between DNS, reverse DNS, or to exit the program. If the user selects the DNS option, the program prompts the user to enter a host name. The program then uses the getByName() method of the InetAddress class to obtain an IP address object for the specified host name. The program outputs the host name and IP address of the specified host name. If the user selects the reverse DNS option, the program prompts the user to enter an IP address. The program then uses the getByName() method of the InetAddress class to obtain a host name object for the specified IP address. The program outputs the IP address and host name of the specified IP address. The program continues to loop until the user selects the exit option.

Output

```
@chirag127 →/workspaces/6th-Sem-CN-Computer-Networks (main) $ cd "/workspaces/6th-Sem-CN-Computer-Networks/practicle/15/" && javac DNS.java && java DNS

Menu:
1. DNS
2. Reverse DNS
3. Exit

Enter your choice
1

Enter Host Name
google.com
Host Name: google.com
IP: 74.125.24.101
```

Conclusion

The program successfully implements DNS (Domain Name System) in Java. It allows the user to enter a host name and displays the corresponding IP address, as well as the ability to enter an IP address and display the corresponding host name.