Department of Computer Science and Engineering

Lab Manual

for

Computer Networks

Regulation 2021

CS3591 – V Semester



ADHIPARASAKTHI ENGINEERING COLLEGE

Melmaruvathur, Chengalpet District- 603319.

CS3591 COMPUTER NETWORKS L T P C 3 0 2 4

COURSE OBJECTIVES:

· To understand the concept of layering in networks.

· To know the functions of protocols of each layer of TCP/IP protocol suite. ·

To visualize the end-to-end flow of information.

· To learn the functions of network layer and the various routing protocols. ·

To familiarize the functions and protocols of the Transport layer.

LIST OF EXPERIMENTS

1. Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and trace route 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: a) Echo client and echo server b) Chat. 4. Simulation of DNS using UDP sockets.
5. Use a tool like Wireshark to capture packets and examine the packets.
6. Write a code simulating ARP /RARP protocols.
7. Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.
8. Study of TCP/UDP performance using Simulation tool.
9. Simulation of Distance Vector/ Link State Routing algorithm.
10. Simulation of an error correction code (like CRC).

TOTAL: 30 PERIODS

COURSE OUTCOMES:

At the end of this course, the students will be able to:

CO 1: Explain the basic layers and its functions in computer networks.

CO 2: Understand the basics of how data flows from one node to another.

CO 3: Analyze routing algorithms.

CO 4: Describe protocols for various functions in the network.

CO 5: Analyze the working of various application layer protocols.

PROGRAM OUTCOMES (POs)

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering Fundamentals and an engineering specialization to the solution of complex engineering Problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex Engineering problems reaching substantiated conclusions using first principles of Mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and Research methods including design of experiments, analysis and interpretation of data, and

synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and Modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to Assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering Solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or Leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the Engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for, and have the preparation and ability to Engage in independent and life-long learning in the broadest context of technological change

PROGRAM SPECIFIC OUTCOMES (PSOs)

The Students will be able to

1. Exhibit design and programming skills to build and automate business solutions using cutting edge technologies.
2. Strong theoretical foundation leading to excellence and excitement towards research, to Provide elegant solutions to complex problems.
3. Ability to work effectively with various engineering fields as a team to design, build and Develop system applications.

CO's-PO's & PSO's MAPPING

CO's	PO's												PSO's		
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
1	-	2	-	-	-	-	-	-	-	-	-	-	3	-	-
2	-	1	-	-	2	-	-	-	-	-	-	2	-	2	-
3	-	2	-	-	3	-	-	-	-	-	-	-	-	3	-
4	-	-	-	1	2	-	-	-	-	3	-	-	-	-	-
5	-	3	2	-	-	-	-	-	-	-	-	-	-	-	3
AVg.	-	1	-	-	1	-	-	-	-	1	-	-	-	1	1

1 - low, 2 - medium, 3 - high, '-"- no correlation

INDEX

Exp.no	Name of the Experiments	Page no
1.	LEARN TO USE COMMANDS LIKE TCPDUMP, NETSTAT, IPCONFIG, NSLOOKUP AND TRACEROUTE. CAPTURE PING AND TRACEROUTE PDUS USING A NETWORK PROTOCOL ANALYZER AND EXAMINE.	2
2.	WRITE A HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP SOCKETS.	7
3.	APPLICATIONS USING TCP SOCKETS LIKE: ECHO CLIENT AND ECHO SERVER CHAT.	11
4.	SIMULATION OF DNS USING UDP SOCKETS.	20
5.	USE A TOOL LIKE WIRESHARK TO CAPTURE PACKETS AND EXAMINE THE PACKETS.	25
6.	WRITE A CODE SIMULATING ARP /RARP PROTOCOLS. PROGRAM FOR ADDRESS RESOLUTION PROTOCOL PROGRAM FOR REVERSE ADDRESS RESOLUTION PROTOCOL.	32
7A.	STUDY OF NETWORK SIMULATOR (NS).	40
7B.	SIMULATION OF CONGESTION CONTROL ALGORITHMS USING NS.	46
8.	STUDY OF TCP/UDP PERFORMANCE USING SIMULATION TOOL.	53
9A.	SIMULATION OF DISTANCE VECTOR ROUTING ALGORITHM.	57
9B.	SIMULATION OF LINK STATE ROUTING ALGORITHM.	61
10.	SIMULATION OF ERROR CORRECTION CODE (LIKE CRC).	66

EX. NO: 1	**LEARN TO USE COMMANDS LIKE TCPDUMP, NETSTAT, IPCONFIG, NSLOOKUP AND TRACEROUTE. CAPTURE PING AND TRACEROUTE PDUS USING A NETWORK PROTOCOL ANALYZER AND EXAMINE**

Aim:

To learn to use commands like tcpdump, netstat, ipconfig, nslookup and traceroute ping.

Commands:

1. Tcpdump

Display traffic between 2 hosts:

To display all traffic between two hosts (represented by variables host1 and host2): # tcpdumphost host1 and host2

Display traffic from a source or destination host only:

To display traffic from only a source (src) or destination (dst) host:

tcpdump src host

tcpdump dst host

Display traffic for a specific protocol

Provide the protocol as an argument to display only traffic for a specific protocol, for example tcp,udp, icmp, arp

tcpdump protocol

For example to display traffic only for the tcp traffic :

tcpdump tcp

Filtering based on source or destination port

To filter based on a source or destination port:

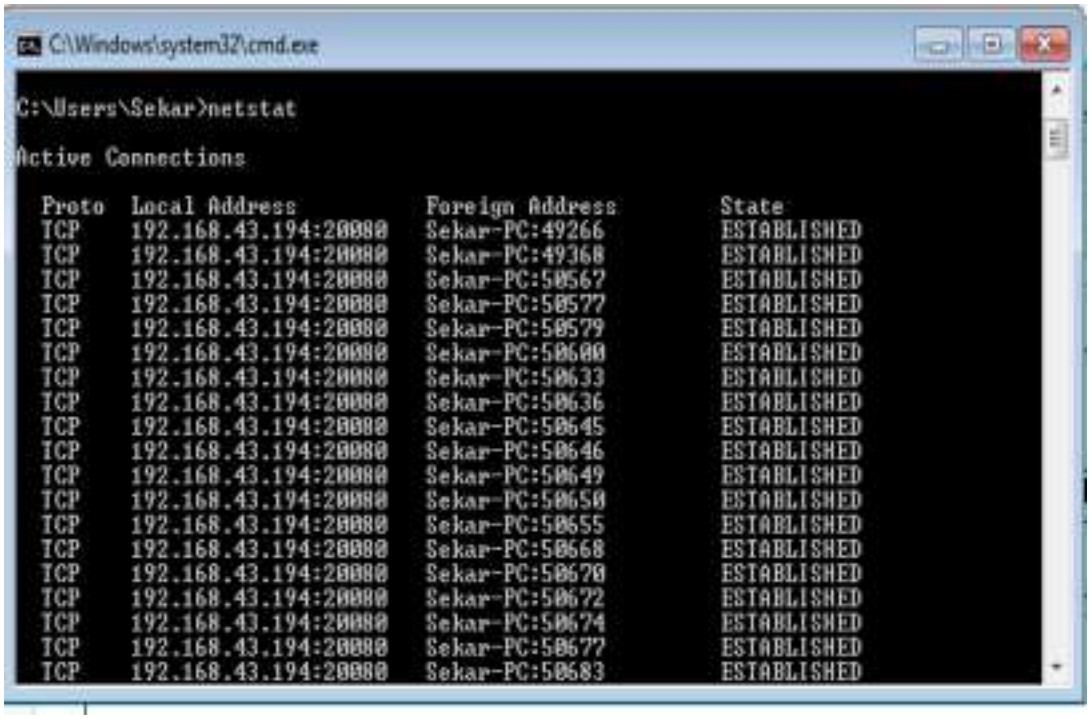
tcpdump src port ftp # tcpdump dst port http

APEC

2. Netstat

Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems. Netstat provides information and statistics about protocols in use and current TCP/IP network connections. The Windows help screen analogous to a Linux or UNIX for netstat reads as follows: displays protocol statistics and current TCP/IP network connections.

#netstat



1. ipconfig

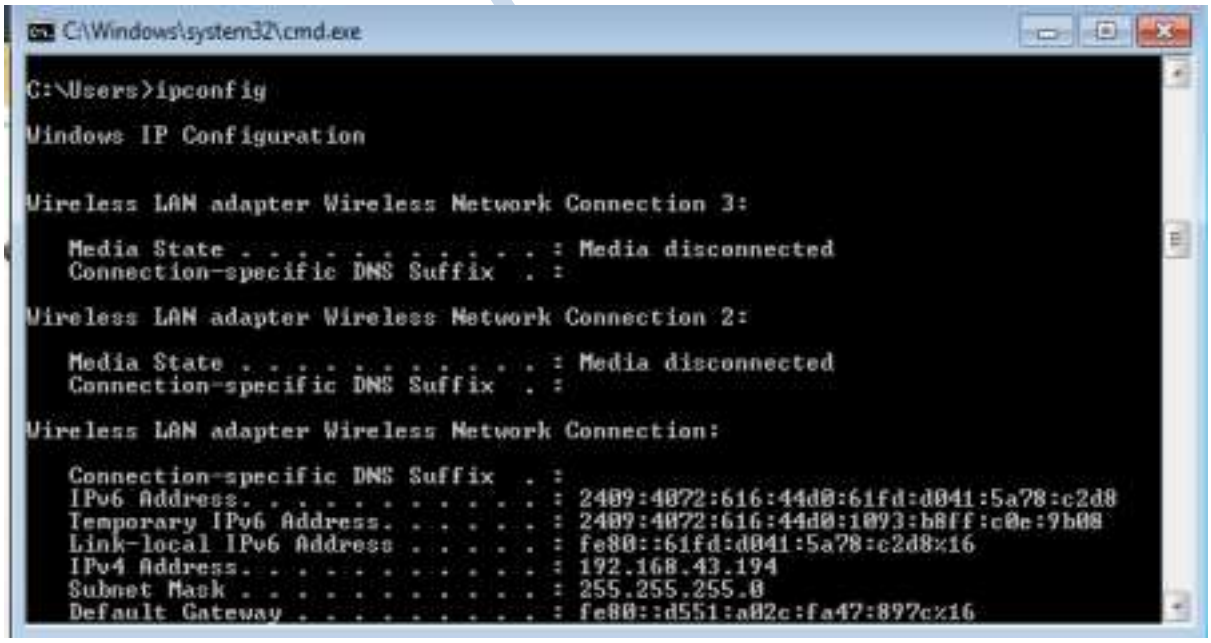
In Windows, **ipconfig** is a console application designed to run from the Windows command prompt. This utility allows you to get the IP address information of a Windows computer.

Using ipconfig

From the command prompt, type **ipconfig** to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter.

APEC

#ipconfig



4. nslookup

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

The nslookup command is a powerful tool for diagnosing DNS problems. You know you're experiencing a DNS problem when you can access a resource by specifying its IP address but not its DNS name.

#nslookup

5. Trace route

Trace route uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop.

Trace route is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Trace route also records the time taken for each hop the packet makes during its route to the destination. Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values.

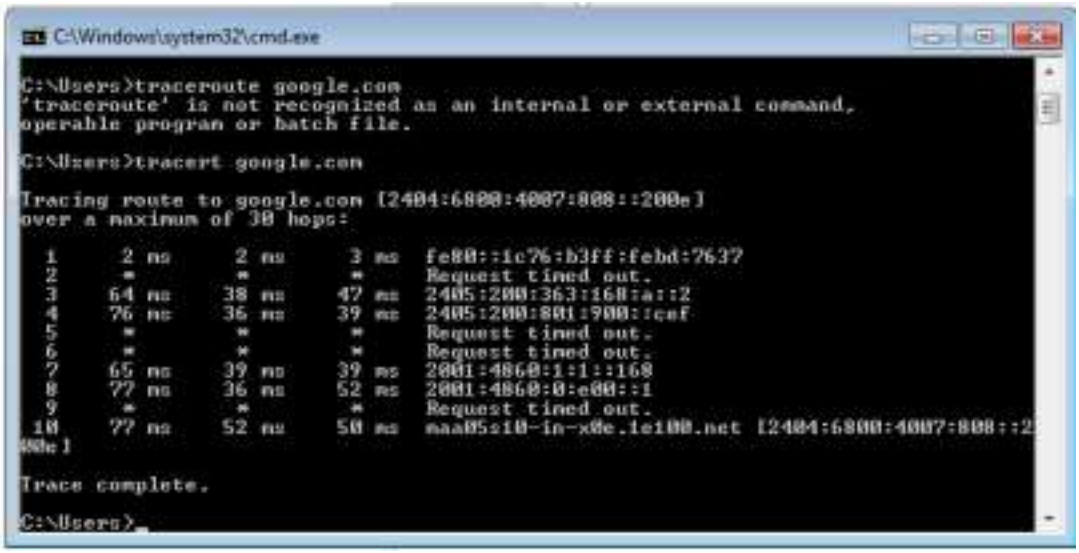
The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop. Trace route sends packets with TTL values that gradually increase from packet to packet, starting with TTL value of one. Routers decrement TTL values of packets by one when routing and discard packets whose TTL value has reached zero, returning the ICMP error message ICMP Time Exceeded.

For the first set of packets, the first router receives the packet, decrements the TTL value and drops the packet because it then has TTL value zero. The router sends an ICMP Time Exceeded message back to the source. The next set of packets are given a TTL value of two, so the first router forwards the packets, but the second router drops them and replies with ICMP Time Exceeded.

Proceeding in this way, trace route uses the returned ICMP Time Exceeded messages to build a list of routers that packets traverse, until the destination is reached and returns an ICMP Echo Reply message.

With the tracert command shown above, we're asking tracert to show us the path from the local computer all the way to the network device with the hostname www.google.com.

#tracert google.com



6. Ping

The ping command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not. Tracking and isolating hardware and software problems. Determine the status of the network and various foreign hosts. The ping command is usually used as a simple way to verify that a computer can communicate over the network with another computer or network device. The ping command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the destination computer and waiting for a response

 # ping172.16.6.2



Result:

Thus, the various networks commands like tcpdump, netstat, ipconfig, nslookup and

traceroute ping are executed successfully.

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

Aim:

To write a java program for socket for HTTP for web page upload and download.

Algorithm:

Client:

1. Start.
2. Create socket and establish the connection with the server.
3. Read the image to be uploaded from the disk
4. Send the image read to the server
5. Terminate the connection
6. Stop.

Server:

1. Start
2. Create socket, bind IP address and port number with the created socket and make server a listening server.
3. Accept the connection request from the client
4. Receive the image sent by the client.
5. Display the image.
6. Close the connection.
7. Stop.

APEC

Program:

Client:

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

public class Client {
 public static void main(String args[]) throws Exception {
 Socket soc;
 BufferedImage img = null;
```



```
 try {
 soc = new Socket("localhost", 4000);
 System.out.println("Client is running.");

 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();

 soc.close();
 } catch (Exception e) {
 System.out.println("Exception: " + e.getMessage());
soc.close();
 }
 }
 }
```

Server:

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

public class Server {
 public static void main(String args[]) throws Exception
{  ServerSocket server = null;
 Socket socket;

 try {
 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();
```

APEC

```
 f.setVisible(true);

 } catch (Exception e) {
 System.out.println("Exception: " + e.getMessage());  }
finally {
 if (server != null) {
 server.close();
 }
 }
 }
}
```


Output:

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



Result:

Thus, the socket program for HTTP for web page upload and download was developed and executed successfully.

EX. NO: 3	APPLICATIONS USING TCP SOCKETS LIKE: ECHO CLIENT AND ECHO SERVER CHAT

Aim:

To write a java program for application using TCP Sockets Links.

a. Echo client and Echo server
b. Chat

Algorithm:

Client:

1. Start.
2. Create the TCP socket.
3. Establish connection with the server.
4. Get the message to be echoed from the user.
5. Send the message to the server.
6. Receive the message echoed by the server.
7. Display the message received from the server.
8. Terminate the connection.
9. Stop.

Server:

1. Start.
2. Create TCP socket, make it a listening socket.
3. Accept the connection request sent by the client for connection establishment.
4. Receive the message sent by the client.
5. Display the received message.
6. Send the received message to the client from which it receives.
7. Close the connection when client initiates termination and server becomes a listening server, waiting for clients.
8. Stop.

Program:

ESERVER:

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



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

```
public class EServer {
 public static void main(String args[]) {
 ServerSocket s = null;
 String line;
 DataInputStream is;
 PrintStream ps;
 Socket c = null;

 try {
 s = new ServerSocket(9000);
 } catch (IOException e) {
 System.out.println(e);
 }

 try {
 System.out.println("Server running. Waiting for client..."); c =
s.accept();
 System.out.println("Client connected.");

 is = new DataInputStream(c.getInputStream()); ps =
new PrintStream(c.getOutputStream());

 while (true) {
```



```
 line = is.readLine();
 ps.println(line);
 }
 } catch (IOException e) {
 System.out.println(e);
 } finally {
 try {
 if (c != null)
```

APEC

```
c.close();
if (s != null)
s.close();
} catch (IOException e)
{  System.out.println(e);  }
}
}
}
```

ECLIENT:

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

public class EClient {
 public static void main(String args[])
{  Socket c = null;
 String line;
 DataInputStream is, is1;
 PrintStream os;

 try {
 InetAddress ia = InetAddress.getLocalHost();  c =
new Socket(ia, 9000);
 } catch (IOException e) {
 System.out.println(e);
 }

 try {
 os = new PrintStream(c.getOutputStream());  is = new
DataInputStream(System.in);  is1 = new
DataInputStream(c.getInputStream());
 while (true) {
 System.out.println("Client:");
```

APEC

```
line = is.readLine();
os.println(line);
System.out.println("Server: " + is1.readLine());  }
} catch (IOException e) {
System.out.println("Socket Closed!");  }
}
}
```


Output:

Server

C:\Program Files\Java\jdk1.5.0\bin>javac EServer.java C:\Program Files\Java\jdk1.5.0\bin>java EServer C:\Program Files\Java\jdk1.5.0\bin> **Client**

C:\Program Files\Java\jdk1.5.0\bin>javac EClient.java C:\Program Files\Java\jdk1.5.0\bin>java EClient

Client: Hai Server

Server: Hai Server

Client: Hello

Server: Hello

Client: end

Server: end

Client: ds

Socket Closed!

APEC

Result:

Thus, the java application program using TCP Sockets for echo client and echo server was developed and executed successfully.

B.Chat

Aim:

To write a java program for application using TCP in Chat.

Algorithm:

1. It uses TCP socket communication. We have a server as well as a client. 2. Both can be run in the same machine or different machines. If both are running in the machine, the address to be given at the client side is local host address.
3. If both are running in different machines, then in the client side we need to specify the ip address of machine in which server application is running.

Program:

Chatserver.java:

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

public class chatserver {
 public static void main(String args[]) throws Exception {
 ServerSocket ss = new ServerSocket(2000);
 Socket sk = ss.accept();

 BufferedReader cin = new BufferedReader(new
InputStreamReader(sk.getInputStream()));
 PrintStream cout = new PrintStream(sk.getOutputStream());
 BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
 String s;

 while (true) {
 s = cin.readLine();
 if (s.equalsIgnoreCase("END")) {
```



```
 cout.println("BYE");
 break;
 }
 System.out.print("Client : " + s + "\n");
 System.out.print("Server : ");
```

APEC

```
 s = stdin.readLine();
 cout.println(s);
 }


 ss.close();
 sk.close();
 cin.close();
 cout.close();
 stdin.close();
 }
}
```

Chatclient.java:

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

public class chatclient {
 public static void main(String args[]) throws Exception {
 Socket sk = new Socket("127.0.0.1", 2000);

 BufferedReader sin = new BufferedReader(new InputStreamReader(sk.getInputStream()));
PrintStream sout = new PrintStream(sk.getOutputStream());
 BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
 String s;

 while (true) {
 System.out.print("Client : ");
 s = stdin.readLine();
 sout.println(s);
 s = sin.readLine();
 System.out.println("Server : " + s);
 if (s.equalsIgnoreCase("BYE")) {
 break;
 }
```



```
 }

 sk.close();
sin.close();
sout.close();
stdin.close();  }
 }
```

APEC

Output:

Server:

E:\nwlab>javac chatserver.java E:\nwlab>java chatserver

Client: hi

Server: hi

Client:

E:\nwlab>javac chatclient.java E:\nwlab>java chatclient

Client: hi

Server: hi

APEC

Result:

Thus, the java application program using TCP Sockets for chat was developed and executed successfully.

EX. NO: 4	SIMULATION OF DNS USING UDP SOCKETS

Aim:

To write a java program for DNS application.

Algorithm:

Server:

1. Start
2. Create UDP datagram socket
3. Create a table that maps host name and IP address
4. Receive the host name from the client
5. Retrieve the client's IP address from the received datagram
6. Get the IP address mapped for the host name from the table.
7. Display the host name and corresponding IP address
8. Send the IP address for the requested host name to the client
9. Stop.

Client:

1. Start
2. Create UDP datagram socket.
3. Get the host name from the client
4. Send the host name to the server
5. Wait for the reply from the server
6. Receive the reply datagram and read the IP address for the requested host name
7. Display the IP address.
8. Stop.

APEC

Program:

DNS Server

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



```
public class udpdnsserver {

 private static int indexOf(String[] array, String str) {
 str = str.trim();
 for (int i = 0; i < array.length; i++) {
 if (array[i].equals(str))
 return i;
 }
 return -1;
 }

 public static void main(String arg[]) throws IOException {
 String[] hosts = {"yahoo.com", "gmail.com", "cricinfo.com", "facebook.com"};  String[]
```

```
ip = {"68.180.206.184", "209.85.148.19", "80.168.92.140", "69.63.189.16"};
 System.out.println("Press Ctrl + C to Quit");

 while (true) {
 DatagramSocket serversocket = new DatagramSocket(1362);  byte[]
senddata = new byte[1021];
 byte[] receivedata = new byte[1021];

 DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);
serversocket.receive(recvpack);

 String sen = new String(recvpack.getData());
 InetAddress ipaddress = recvpack.getAddress();
 int port = recvpack.getPort();
 String capsent;

 System.out.println("Request for host " + sen);
```



```
 if (indexOf(hosts, sen) != -1)
 capsent = ip[indexOf(hosts, sen)];
 else
 capsent = "Host Not Found";

 senddata = capsent.getBytes();
 DatagramPacket pack = new DatagramPacket(senddata, senddata.length, ipaddress,  port);
 serversocket.send(pack);
 serversocket.close();
 }
 }
}
```

DNS Client

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

public class udpdnsclient {
 public static void main(String args[]) throws IOException {
 BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
DatagramSocket clientsocket = new DatagramSocket();
```

APEC

```
InetAddress ipaddress;

if (args.length == 0)
ipaddress = InetAddress.getLocalHost();
else
ipaddress = InetAddress.getByName(args[0]);

int portaddr = 1362;
byte[] senddata = new byte[1024];
byte[] receivedata = new byte[1024];
```



```
System.out.print("Enter the hostname : ");
String sentence = br.readLine();
senddata = sentence.getBytes();

DatagramPacket pack = new DatagramPacket(senddata, senddata.length, ipaddress,
portaddr);
clientsocket.send(pack);

DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);
clientsocket.receive(recvpack);

String modified = new String(recvpack.getData());
System.out.println("IP Address: " + modified.trim());

clientsocket.close();
}
}
```

APEC

Output:

Server:

javac udpdnsserver.java java udpdnsserver

Press Ctrl + C to Quit Request for host yahoo.com Request for host

cricinfo.com Request for host youtube.com

Client:

Javac udpdnsclient.java

Java udpdnsclient

Enter the hostname: yahoo.com IP Address: 68.180.206.184

java udpdnsclient

Enter the hostname: cricinfo.com IP Address: 80.168.92.140

java udpdnsclient

Enter the hostname: youtube.com IP Address: Host Not Found

APEC

Result:

Thus, the java application program using UDP Sockets to implement DNS was developed and executed successfully

EX. NO: 5	USE A TOOL LIKE WIRESHARK TO CAPTURE PACKETS AND EXAMINE THE PACKETS

Wireshark is an open-source network protocol analysis software program, widely considered the industry standard. A global organization of network specialists and software developers supports Wireshark and continues to make updates for new network technologies and

encryption methods.

When should Wireshark be used?

Wireshark can be used to understand how communication takes place across a network and to analyse what went wrong when an issue in communication arises. It helps · Network administrators troubleshoot problems across a network

- Security engineers examine security issues across a network
- QA engineers verify applications
- Developers debug protocol implementations
- Network users learn about a specific protocol

Installing Wireshark

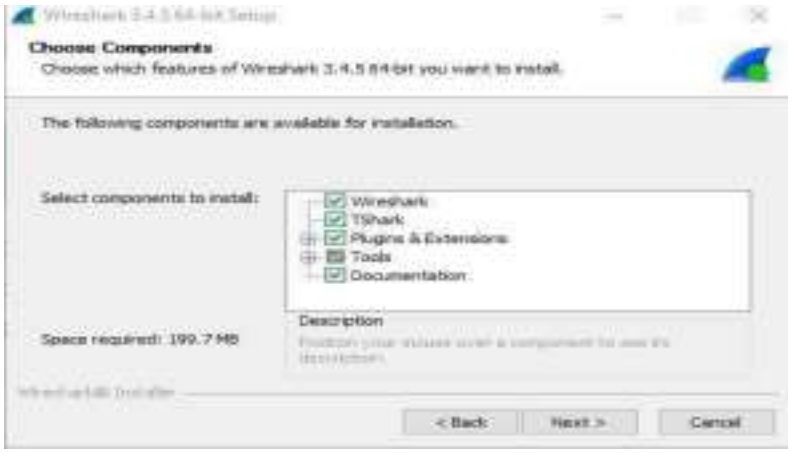
Go to https://www.wireshark.org/download.html to visit the Wireshark download page. Ensure To use the full URL. The output will look similar to the following



Download the Windows Installer (64 bit).

Click on the link for your version of software, and you will see a pop-up box at the foot of your screen.

Choose **run**, and when the **UAC prompt** appears, choose **yes**. This will bring up the installation wizard.



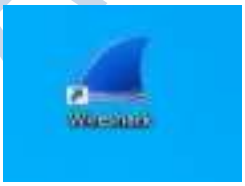
Keep pressing 'Next' and accept the defaults. The installation will commence, and the pop-up box below will appear. **Accept** the license agreement and press **I Agree**. Accept the default settings.



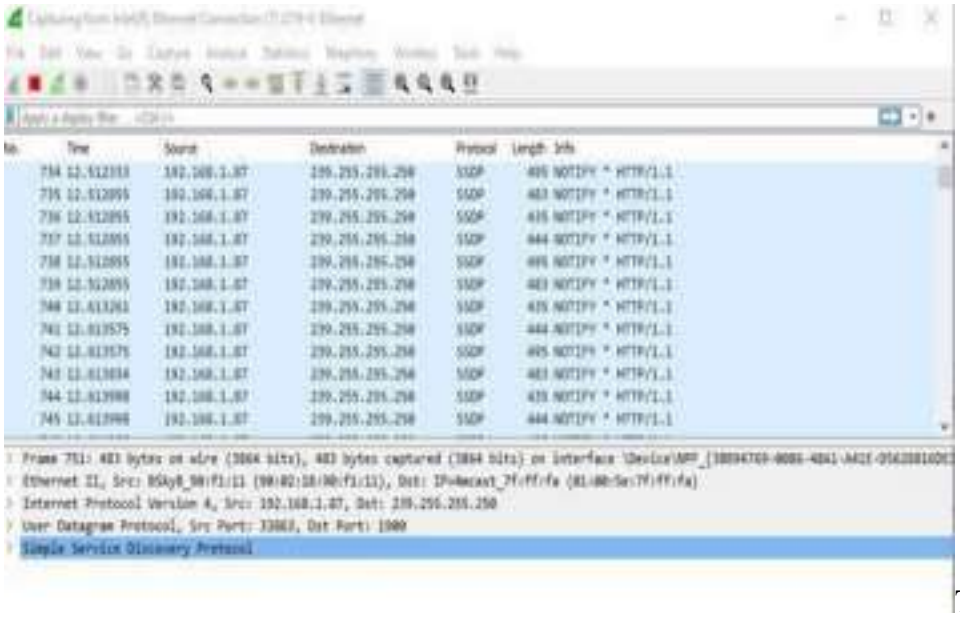
Accept the default settings by pressing **Next**. The following wizard will appear (see below). Press **Finish**.



The Wireshark installation will still be running in the background. It should take roughly another 2-3 minutes. The wizard will appear to say the installation is complete. Select **Next**, then **Finish**. You will now see a Wireshark shortcut on the desktop, the same as below:



Double-click it and choose your network interface. When your Wireshark console appears, it should look similar to that shown below. If you need to change the interface, go to **Capture** and select **Options**.



The view above shows **The Main Window**, which is broken into different sections: **The Menu**: This is broken into the following 11 headings: File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help. **The Filter Toolbar**: This has a filter pane when you type in the protocol that you want to view. **The Packet List**: This shows all packets that are captured and is shown in blue in the preceding image. **The Packet Details Pane**: This is the gray area that shows the protocol fields of the packet. **The Packet Bytes Pane**: This shows a canonical hex dump of the packet data.

APEC

How to Capture Data Packets

One of the core functions of Wireshark as a network analysis tool is to capture packets of data. Before you start to capture packets, there are three things you need to do: **Make sure that you have the administrative privileges** to start a live capture on your device

Choose the correct network interface to capture packet data from

Capture packet data **from the correct location** in your network

Once these three things are done, you're ready to start the capture process. When you use Wireshark to capture packets, they are displayed in a human-readable format to make them legible to the user. You can also **break packets down with filters and color-coding** if you wish to see more specific information. When you first open up Wireshark, you'll be met by the following launch screen:

The first thing you need to do is look at the available interfaces to capture. To do this, select **Capture > Options**. The "Capture Interfaces" dialog box will then open as shown below:

APEC

Check the box of the interface you want to capture and press the **Start** button to start. You can select multiple interfaces if you want to capture data from multiple sources simultaneously. On Unix or Linux, the dialog box is shown in a similar style like this:

Promiscuous Mode

Promiscuous mode is **an interface mode where Wireshark details every packet it sees**. When this mode is deactivated, you lose transparency over your network and only develop a limited snapshot of your network (this makes it more difficult to conduct any analysis). To activate promiscuous mode, click on the **Capture Options** dialog box and click **promiscuous mode**.

In theory, this should show you all the traffic active on your network. The promiscuous mode box is shown below:

However, this often isn't the case. Many network interfaces are resistant to promiscuous mode, so you need to check the Wireshark website for information on your specific hardware. On Windows, it's useful to open **Device Manager** and check whether you have your settings configured to reject promiscuous mode. For example:

APEC

(Simply click on network and then make sure that your promiscuous mode setting are set to **Allow All).**

If you have your settings set to "reject" promiscuous mode, then you're going to limit the number of packets Wireshark captures. So even if you have promiscuous mode enabled on Wireshark check your Device Manager to make sure that your interface isn't blocking any data from coming through.

Taking the time to check through your network infrastructure will ensure Wireshark receives all the necessary packets of data.

How to Examine Captured Packets

Once you've captured your network data, you'll want to look at your captured packets. In the screenshot below you'll see three panes, the **packet list** pane, the **packet bytes** pane, and the **packet details** pane.

If you want more information, you can click on any of the fields in each packet to see more. When you click on a packet, you're shown a breakdown of its internal bytes in the byte view section.

Packet List

The packet list pane is shown at the top of the screenshot. Each piece is broken down to a

number with time, source, destination, protocol and support information. **Packet Details**

Packet details can be found in the middle, showing the protocols of the chosen packet. You can expand each section by clicking on the arrow next to your row of choice. You can also apply additional filters by right-clicking on the chosen item.

Packet Bytes

The packet bytes pane is shown at the bottom of the page. This pane shows the internal data of your selected packet. If you highlight part of the data in this section, its corresponding information is also highlighted in the packet details pane. By default, all data is shown in hexadecimal format. If you want to change it to bit format, right-click the pane and select this option from the context menu.

Result:

Thus, Wireshark tool is used to Capture packets and examine the packets.

EX. NO: 6	WRITE A CODE SIMULATING ARP /RARP PROTOCOLS. PROGRAM FOR ADDRESS RESOLUTION PROTOCOL PROGRAM FOR REVERSE ADDRESS RESOLUTION PROTOCOL.

(A)Program for Address Resolution Protocol (ARP) using TCP

Aim:

To write a java program for simulating ARP and protocols using TCP.

Algorithm:

Client:

1. Start the program.
2. Create socket and establish connection with the server.
3. Get the IP address to be converted into MAC address from the user.
4. Send this IP address to server.
5. Receive the MAC address for the IP address from the server.
6. Display the received MAC address.
7. Terminate the connection.

Server:

1. Start the program.
2. Create the socket, bind the socket created with IP address and port number and make it a listening socket.
3. Accept the connection request when it is requested by the client.
4. Server maintains the table in which IP and corresponding MAC addresses are stored.
5. Receive the IP address sent by the client.
6. Retrieve the corresponding MAC address for the IP address and send it to the client.
7. Close the connection with the client and now the server becomes a listening server waiting for the connection request from other clients.
8. Stop.

Program:

Client:

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



```
class Clientarp {
 public static void main(String args[]) {
 try {
 BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
Socket clsct = new Socket("127.0.0.1", 139);

 DataInputStream din = new DataInputStream(clsct.getInputStream());
DataOutputStream dout = new DataOutputStream(clsct.getOutputStream());
 System.out.println("Enter the Logical address(IP):");
 String str1 = in.readLine();
 dout.writeBytes(str1 + '\n');

 String str = din.readLine();
 System.out.println("The Physical Address is: " + str);

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

Server:

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

class Serverarp {
 public static void main(String args[]) {
 try {
 ServerSocket obj = new ServerSocket(139);
 Socket obj1 = obj.accept();

 while (true) {
 DataInputStream din = new DataInputStream(obj1.getInputStream());
DataOutputStream dout = new DataOutputStream(obj1.getOutputStream());
 String str = din.readLine();
```

```
String ip[] = {"165.165.80.80", "165.165.79.1"};
String mac[] = {"6A:08:AA:C2", "8A:BC:E3:FA"};

for (int i = 0; i < ip.length; i++) {
if (str.equals(ip[i])) {
dout.writeBytes(mac[i] + '\n');
break;
```



```
}
}

obj.close(); // Note: Closing the server socket inside the loop might not be intended  }
} catch (Exception e) {
System.out.println(e);
}
}
}
```

APEC

Output:

E:\networks>java Serverarp E:\networks>java Clientarp Enter the Logical address (IP): 165.165.80.80

The Physical Address is: 6A:08: AA:C2

APEC

Result:

Thud, the java program for simulating ARP protocols using TCP was executed and verified successfully.

(B) Program for Reverse Address Resolution Protocol (RARP) using UDP Aim:

To write a java program for simulating RARP and protocols using UDP.

Algorithm:

Client:

1. Start the program
2. Create datagram socket

3. Get the MAC address to be converted into IP address from the user.
4. Send this MAC address to server using UDP datagram.
5. Receive the datagram from the server and display the corresponding IP address.
6. Stop

Server:

1. Start the program.
2. Server maintains the table in which IP and corresponding MAC addresses are stored.
3. Create the datagram socket
4. Receive the datagram sent by the client and read the MAC address sent.
5. Retrieve the IP address for the received MAC address from the table. 6.

Display the corresponding IP address.

7. Stop

Program:

Client:

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

class Clientrarp12 {
 public static void main(String args[]) {
 try {
 DatagramSocket client = new DatagramSocket();
 InetAddress addr = InetAddress.getByName("127.0.0.1");
 byte[] sendbyte = new byte[1024];
 byte[] receivebyte = new byte[1024];

 BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```



```
 System.out.println("Enter the Physical address (MAC):");
 String str = in.readLine();
 sendbyte = str.getBytes();

 DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length, addr, 1309);
client.send(sender);

 DatagramPacket receiver = new DatagramPacket(receivebyte, receivebyte.length);
client.receive(receiver);

 String s = new String(receiver.getData());
 System.out.println("The Logical Address is(IP): " + s.trim());

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

Server:

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

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

class Serverrarp12 {
 public static void main(String args[]) {
 try {
 DatagramSocket server = new DatagramSocket(1309);

 while (true) {
 byte[] sendbyte = new byte[1024];
 byte[] receivebyte = new byte[1024];

 DatagramPacket receiver = new DatagramPacket(receivebyte, receivebyte.length);
server.receive(receiver);

 String str = new String(receiver.getData());
 String s = str.trim();
```



```
 InetAddress addr = receiver.getAddress();
 int port = receiver.getPort();

 String ip[] = {"165.165.80.80", "165.165.79.1"};
 String mac[] = {"6A:08:AA:C2", "8A:BC:E3:FA"};

 for (int i = 0; i < ip.length; i++) {
 if (s.equals(mac[i])) {
 sendbyte = ip[i].getBytes();
 DatagramPacket sender = new DatagramPacket(sendbyte, sendbyte.length,  addr, port);
 server.send(sender);
 break; // Exit the loop after sending the response
 }
 }
 }
 } catch (Exception e) {
 System.out.println(e);
 }
 }
}
```

APEC

Output:

I:\ex>java Serverrarp12 I:\ex>java Clientrarp12

Enter the Physical address (MAC):

6A:08: AA:C2

The Logical Address is (IP): 165.165.80.80

APEC

Result:

Thus, the program for implementing to display simulating RARP protocols was executed successfully and output is verified.

EX. NO: 7A	STUDY OF NETWORK SIMULATOR (NS).

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 behaviours. 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 Défense 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

APEC

Figure 2.1 shows the basic architecture of NS2. NS2 provides users with 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 commands.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++ definesthe 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 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. Itdefines 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 are encouraged 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.2, respectively.

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 aOTcl 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 requires 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, andMac.

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 seta 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.

Initialization and termination aspects of the simulator.

Definition of network nodes, links, queues and topology.

Definition of agents and of applications.

The nam visualization tool.

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

Define a "finish" procedure Proc finish { } { global ns tracefile1 namfile

$ns flush-trace Close $tracefile1 Close $namfile Exec namout.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 destinationto know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agentappears 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 andassigns 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 has been studied in detail.

EX. NO: 7B	SIMULATION OF CONGESTION CONTROL ALGORITHMS USING NS

CONGESTION CONTROL

Congestion refers to a network state where a node or link carries so much data that it may deteriorate network service quality, resulting in queuing delay, frame or data packet loss and the blocking of new connections.

In Congestion control, end systems throttle back in order to avoid congesting the network. The mechanism is similar to end-to-end flow controls, but the intention is to reduce congestion in the network, not the receiver.

Network simulator 2 (Ns2 Program for congestion control outputs better results).

Techniques available in congestion control:

Open Loop Technique and closed Loop Technique are utilized in ***ns2 program for congestion control.***

Open loop

- Acknowledgement policy.
- Retransmission policy.
- Discarding policy.
- Window policy.
- Admission policy.

Closed loop

- Explicit feedback.
- Back pressure.
- Implicit feedback.
- Choke packet.

APEC

TCL SCRIPT FOR CONGESTION CONTROL IN TCP

```
# Create a new simulator instance
set ns [new Simulator]

# Open trace and nam files for output
set f [open congestion.tr w]
$ns trace-all $f

set nf [open congestion.nam w]
$ns namtrace-all $nf

# Define colors for nodes
$ns color 1 Red
$ns color 2 Blue
$ns color 3 White
$ns color 4 Green

# Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
```



```
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

# Create links between nodes with bandwidth, delay, and queue type
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.3Mb 200ms DropTail
$ns duplex-link $n3 $n4 0.5Mb 40ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 30ms DropTail
```

APEC

```
# Sending nodes with Reno TCP agents
set tcp1 [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp1

set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp2

set tcp3 [new Agent/TCP/Reno]
$ns attach-agent $n2 $tcp3

set tcp4 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp4

# Set flow IDs for TCP agents
$tcp1 set fid_ 1
$tcp2 set fid_ 2
$tcp3 set fid_ 3
$tcp4 set fid_ 4

# Receiving (sink) nodes
set sink1 [new Agent/TCPSink]
```



```
$ns attach-agent $n4 $sink1

set sink2 [new Agent/TCPSink]
$ns attach-agent $n5 $sink2

set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3

set sink4 [new Agent/TCPSink]
$ns attach-agent $n4 $sink4

# Connect TCP agents with sinks
$ns connect $tcp1 $sink1 $ns
connect $tcp2 $sink2 $ns
connect $tcp3 $sink3 $ns
connect $tcp4 $sink4

# Setup FTP traffic generators
```

APEC

```
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP

set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$ftp3 set type_ FTP

set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp4
```



```
$ftp4 set type_ FTP

# Setup Ping agents
set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0

set p1 [new Agent/Ping]
$ns attach-agent $n4 $p1

# Connect Ping agents
$ns connect $p0 $p1

# Schedule ping sends
$ns at 0.2 "$p0 send"
$ns at 0.3 "$p1 send"

# Start FTP traffic
$ns at 0.5 "$ftp1 start"
$ns at 0.6 "$ftp2 start"
$ns at 0.7 "$ftp3 start"
$ns at 0.8 "$ftp4 start"

# Stop FTP traffic
$ns at 66.0 "$ftp4 stop"
$ns at 67.0 "$ftp3 stop"
```

APEC

```
$ns at 68.0 "$ftp2 stop"
$ns at 70.0 "$ftp1 stop"

# Schedule additional ping
sends $ns at 70.1 "$p0 send"
$ns at 70.2 "$p1 send"
```



```
# Set simulation end time
$ns at 80.0 "finish"

# Procedure to plot the congestion window (cwnd_)
proc plotWindow {tcpSource outfile} {
 global ns
 set now [$ns now]
 set cwnd_ [$tcpSource set cwnd_]
 puts $outfile "$now $cwnd_"
 $ns at [expr $now + 0.1] "plotWindow $tcpSource $outfile" }

# Open file for congestion window plotting
set outfile [open "congestion.xg" w]

# Schedule initial plotWindow call
$ns at 0.0 "plotWindow $tcp1 $outfile"

# Procedure to finish simulation
proc finish {} {
 exec nam congestion.nam &
 exec xgraph congestion.xg -geometry 300x300 &
exit 0
}

# Run simulation
$ns run
```


Output:

APEC

Result:

Thus, the Congestion Control Algorithm has been Simulated and studied.

EX. NO: 8	STUDY OF TCP/UDP PERFORMANCE USING SIMULATION TOOL

Aim:

To simulate the study of TCP and UDP using NS2.

TRANSMISSION CONTROL PROTOCOL

In theory, a transport layer protocol could be a very simple software routine, but the TCP protocol cannot be called simple. Why use a transport layer which is as complex as TCP? The most important reason depends on IP's unreliability. In fact all the layers below TCP are

unreliable and deliver the datagram hop by- hop. The IP layer delivers the datagram hop-by hop and does not guarantee delivery of a datagram; it is a connectionless system. IP simply handles the routing of datagram's; and if problems occur, IP discards the packet without a second thought, generating an error message back to the sender in the process. The task of ascertaining the status of the datagram sent over a network and handling the resending of information if parts have been discarded falls to TCP. TCP manages the flow of datagrams from the higher layers, as well asincoming datagrams from the IP layer. It has to ensure that priorities and security are respected. TCP must be capable of handling the termination of an application above it that was expecting incoming datagrams, as well as failures in the lower layers. TCP also must maintain a state table of all data streams in and out of the TCP layer. The isolation of these services in a separate layer enables applications to be designed without regard to flow control or message reliability. Without the TCP layer, each application would have to implement the services themselves, which is a waste of resources.

TCP resides in the transport layer, positioned above IP but below the upper layers and their applications. TCP resides only on devices that actually process datagram's, ensuring that the datagram has gone from the source to target machines. It does not reside on a device that simply routes datagrams, so there is no TCP layer in a gateway. This makes sense, because on a gateway the datagram has no need to go higher in the layered model than the IP layer.

User Datagram Protocol:

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



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

proc finish {} {
 global ns nf
 $ns flush-trace
 close $nf
 exec nam out.nam &
 exit 0
}

set h1 [$ns node]
set r1 [$ns node]
set h2 [$ns node]
set h3 [$ns node]
set r2 [$ns node]
set h4 [$ns node]
```

APEC

```
$ns duplex-link $h1 $r1 10Mb 20ms DropTail $ns duplex-link $h2 $r1 10Mb 20ms DropTail $ns duplex-link $r1 $r2 1.5Mb 20ms DropTail $ns duplex-link $r2 $h3 10Mb 20ms DropTail $ns duplex-link $r2 $h4 5Mb 20ms DropTail

$ns duplex-link-op $h1 $r1 orient right-down $ns duplex-link-op $h2 $r1 orient right-up $ns duplex-link-op $r1 $r2 orient right $ns duplex-link-op $r2 $h3 orient right-up $ns duplex-link-op $r2 $h4 orient right-down

set udp [new Agent/UDP]
```



```
$ns attach-agent $h2 $udp

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

set null [new Agent/Null]
$ns attach-agent $h4 $null

set tcp [new Agent/TCP]
$ns attach-agent $h1 $tcp

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

set sink [new Agent/TCPSink]
$ns attach-agent $h3 $sink

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

$ns at 0.0 "$cbr start"
$ns at 0.0 "$ftp start"
$ns at 10.0 "finish"
```

APEC

$ns run

Output:

APEC

Result:

Thus, the NS2 with TCP and UDP packets has been Simulated and studied.

EX. NO: 9A	SIMULATION OF DISTANCE VECTOR ROUTING ALGORITHM

Aim:

To simulate and study the Distance Vector routing algorithm using

simulation. **Software required:** NS-2

Theory:

Distance Vector Routing is one of the routing algorithms in a Wide Area Network for computing shortest path between source and destination. The Router is one main device used in a wide area network. The main task of the router is Routing. It forms the routing table and delivers the packets depending upon the routes in the table- either directly or via an intermediate device. Each router initially has information about its all neighbors. Then this information will be shared among nodes.

Algorithms:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on tracefile.
4. Create n number of nodes using 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 distance vector routing protocol to transmit data from sender to receiver. 10. Schedule events and run the program.

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 ns nf 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
```

APEC

```
$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
```


Output:

APEC

Result:

Thus, the Distance vector Routing Algorithm has been Simulated and studied.

EX. NO: 9B	SIMULATION OF LINK STATE ROUTING ALGORITHM.

Aim:

To simulate and study the link state routing algorithm using simulation.

Software Required: NS-2

Theory:

In link state routing, each router shares its knowledge of its neighbourhood with every other router in the internetwork. (i) Knowledge about Neighbourhood: Instead of sending its entire routing table a router sends info about its neighborhood only. (ii) To all Routers: each router sendsthisinformation to every other router on the internetwork not just to its neighbor. It does so by a process called flooding. (iii)Information sharing when there is a change: Each router sends out information about the neighbors when there is change.

Procedure:

The Dijkstra algorithm follows four steps to discover what is called the shortest path tree (routing table) for each router: The algorithm begins to build the tree by identifying its roots. The root router's trees the router itself. The algorithm then attaches all nodes that can be reached from the root. The algorithm compares the tree's temporary arcs and identifies the arc with the lowest cumulative cost. This arc and the node to which it connects are now a permanent part of the shortest path tree. The algorithm examines the database and identifies every node that can be reached from its chosen node. These nodes and their arcs are added temporarily to the tree. The last two steps are repeated until every node in the network has become a permanent part of the tree.

Algorithms:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on tracefile.

4. Create n number of nodes using 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:

```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr

set nf [open thro.nam w]
$ns namtrace-all $nf

proc finish {} {
 global ns nr nf
 $ns flush-trace
 close $nf
 close $nr
 exec nam thro.nam &
 exit 0
}

# Create nodes
for {set i 0} {$i < 12} {incr i} {
 set n($i) [$ns node]
}

# Create duplex links with specified parameters
```



```
for {set i 0} {$i < 8} {incr i} {
 $ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }
$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms
DropTail $ns duplex-link $n(11) $n(5) 1Mb 10ms
DropTail
```

APEC

```
# Create UDP agents and attach to nodes
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1

set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

set null1 [new Agent/Null]
$ns attach-agent $n(5)
$null1 $ns connect $udp1
$null1

# Routing protocol setup
$ns rtproto LS
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)

# Set fid (Flow ID) for UDP
agents $udp0 set fid_ 1
$udp1 set fid_ 2

# Node colors for visualization
$ns color 1 Red
```

APEC

```
$ns color 2 Green

# Start traffic at specified times
$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"

# Schedule simulation end
$ns at 45 "finish"

# Run the simulation
$ns run
```


Output:

APEC

Result:

Thus, the Link State Routing Algorithm has been Simulated and studied.

EX. NO: 10	SIMULATION OF ERROR CORRECTION CODE (LIKE CRC)

Aim:

To implement error checking code using java.

Algorithm:

1. Start the Program
2. Given a bit string, append 0S to the end of it (the number of 0s is the same as the degree of the generator polynomial) let B(x) be the polynomial corresponding to B. 3. Divide B(x) by some agreed on polynomial G(x) (generator polynomial) and determine the remainder R(x). This division is to be done using Modulo 2 Division. 4. Define T(x) = B(x) –R(x)
5. (T(x)/G(x) => remainder 0)
6. Transmit T, the bit string corresponding to T(x).
7. Let T' represent the bit stream the receiver gets and T'(x) the associated polynomial. The receiver divides T1(x) by G(x). If there is a 0 remainder, the receiver concludes T = T' and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.
8. Stop the Program

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];
```

APEC

```
}
rem = divide(div, divisor, rem);

for (int i = 0; i < div.length; i++) {
// append dividend and remainder
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 :)");  }
```

APEC

```
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;
}
}
```

APEC

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
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
THANK YOU:)
BUILD SUCCESSFUL (total time: 1 minute 34 seconds)
```


Result:

Thus, the above program for error checking code using was executed successfully.

APEC

APEC