M.E.T ENGINEERING COLLEGE

Mogals garden, Chenbagaramanputhoor, 629304.

Register No:

PRACTICAL RECORD

Certified that the Bonafide Record work is done by Selvan / Selvi ....

.... of .... Semester of ....

Engineering during the year...................... ....in the ....

.. laboratory.

Staff-in-Charge **Head of the Department**

Submitted for the Anna University, Chennai, Practical Examination held on

*................... at **M.E.T Engineering College, Chenbagaramanputhoor.***

Internal Examiner **External Examiner**

INDEX

EX NO: 1 **Basic Commands**

DATE:

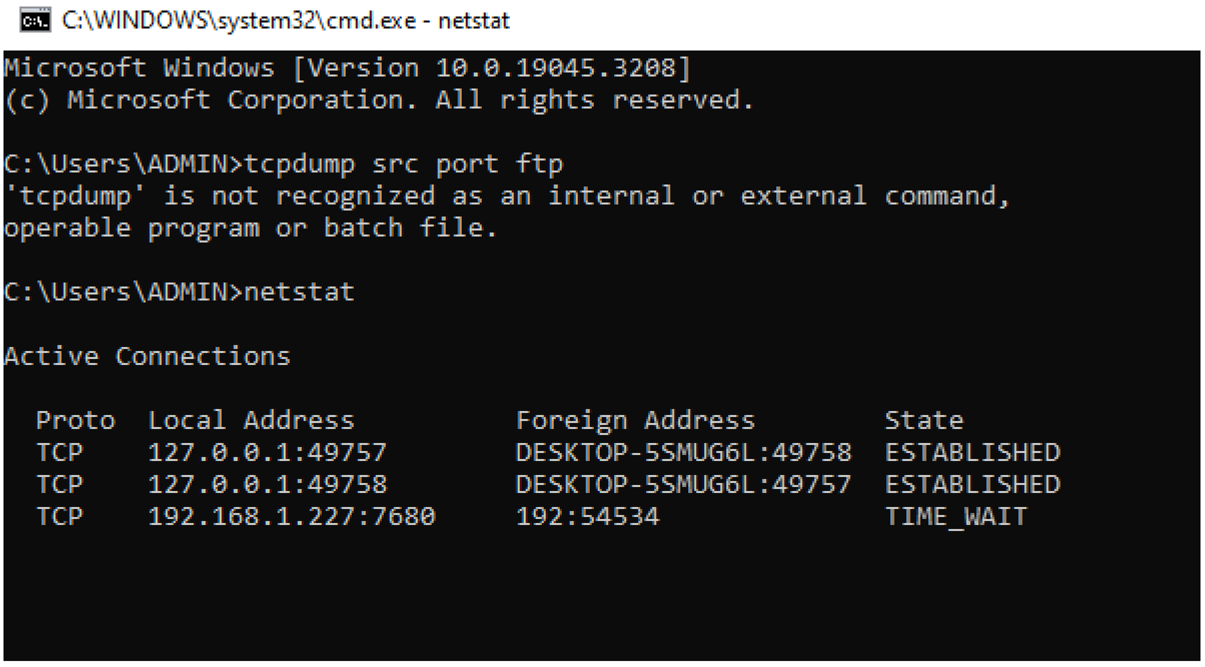
AIM:

Learn to use commands like tcpdump, netstat, ifconfig, nslookup and tracert. Capture ping and trace route PDUs using a network protocol analyzer and examine.

Commands:

1. **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.



```
C:\WINDOWS\system32\cmd.exe - netstat
Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>tcpdump src port ftp
'tcpdump' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\ADMIN>netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    127.0.0.1:49757        DESKTOP-5SMUG6L:49758  ESTABLISHED
  TCP    127.0.0.1:49758        DESKTOP-5SMUG6L:49757  ESTABLISHED
  TCP    192.168.1.227:7680     192:54534              TIME_WAIT
```

2. **TCPDUMP:**

Display traffic between 2 hosts:

- To display all traffic between two hosts (represented by variables host1 and host2): # tcpdump host 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.

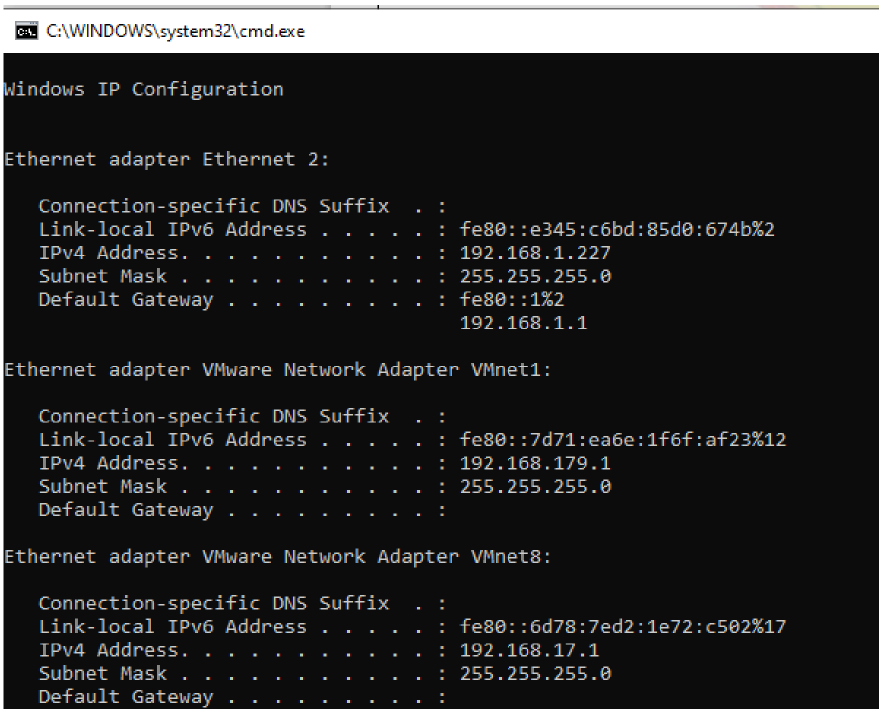
3. Ipconfig:

- 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.

  #ipconfig

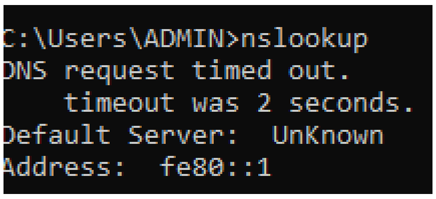


```
Windows IP Configuration


Ethernet adapter Ethernet 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::e345:c6bd:85d0:674b%2
   IPv4 Address. . . . . . . . . . . : 192.168.1.227
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::1%2
                                       192.168.1.1

Ethernet adapter VMware Network Adapter VMnet1:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::7d71:ea6e:1f6f:af23%12
   IPv4 Address. . . . . . . . . . . : 192.168.179.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Ethernet adapter VMware Network Adapter VMnet8:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::6d78:7ed2:1e72:c502%17
   IPv4 Address. . . . . . . . . . . : 192.168.17.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

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.

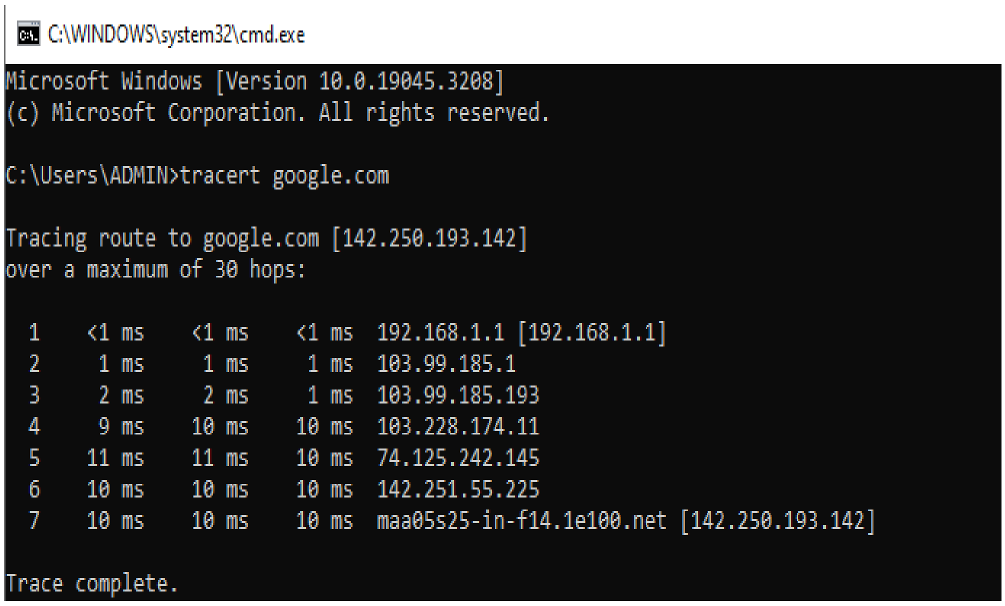
#nslookup



```
:\Users\ADMIN>nslookup
NS request timed out.
    timeout was 2 seconds.
efault Server:  UnKnown
ddress:  fe80::1
```

5. Trace route:

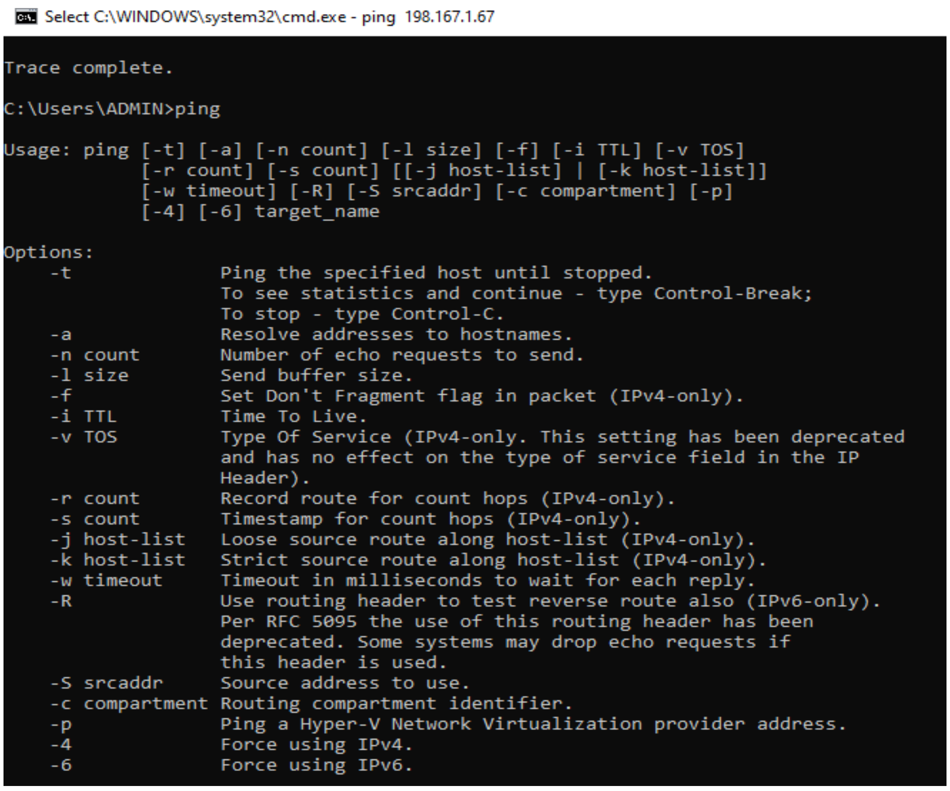
- Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop.
- Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute 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. Traceroute 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, traceroute 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.



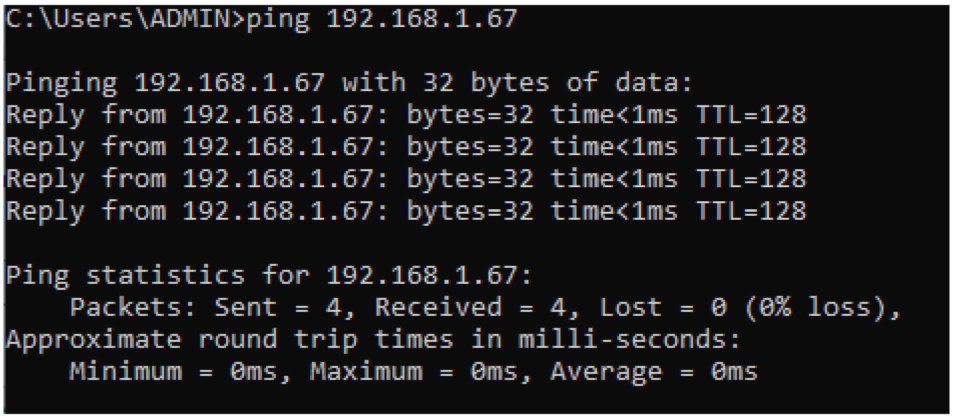
```
C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Version 10.0.19045.3208]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>tracert google.com

Tracing route to google.com [142.250.193.142]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  192.168.1.1 [192.168.1.1]
  2     1 ms     1 ms     1 ms  103.99.185.1
  3     2 ms     2 ms     1 ms  103.99.185.193
  4     9 ms    10 ms    10 ms  103.228.174.11
  5    11 ms    11 ms    10 ms  74.125.242.145
  6    10 ms    10 ms    10 ms  142.251.55.225
  7    10 ms    10 ms    10 ms  maa05s25-in-f14.1e100.net [142.250.193.142]

Trace complete.
```

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. Determining 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.



```
Select C:\WINDOWS\system32\cmd.exe - ping  198.167.1.67

Trace complete.

C:\Users\ADMIN>ping

Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS]
            [-r count] [-s count] [[-j host-list] | [-k host-list]]
            [-w timeout] [-R] [-S srcaddr] [-c compartment] [-p]
            [-4] [-6] target_name

Options:
    -t             Ping the specified host until stopped.
                   To see statistics and continue - type Control-Break;
                   To stop - type Control-C.
    -a             Resolve addresses to hostnames.
    -n count       Number of echo requests to send.
    -l size        Send buffer size.
    -f             Set Don't Fragment flag in packet (IPv4-only).
    -i TTL         Time To Live.
    -v TOS         Type Of Service (IPv4-only. This setting has been deprecated
                   and has no effect on the type of service field in the IP
                   Header).
    -r count       Record route for count hops (IPv4-only).
    -s count       Timestamp for count hops (IPv4-only).
    -j host-list   Loose source route along host-list (IPv4-only).
    -k host-list   Strict source route along host-list (IPv4-only).
    -w timeout     Timeout in milliseconds to wait for each reply.
    -R             Use routing header to test reverse route also (IPv6-only).
                   Per RFC 5095 the use of this routing header has been
                   deprecated. Some systems may drop echo requests if
                   this header is used.
    -S srcaddr     Source address to use.
    -c compartment Routing compartment identifier.
    -p             Ping a Hyper-V Network Virtualization provider address.
    -4             Force using IPv4.
    -6             Force using IPv6.
```



```
C:\Users\ADMIN>ping 192.168.1.67

Pinging 192.168.1.67 with 32 bytes of data:
Reply from 192.168.1.67: bytes=32 time<1ms TTL=128
Reply from 192.168.1.67: bytes=32 time<1ms TTL=128
Reply from 192.168.1.67: bytes=32 time<1ms TTL=128
Reply from 192.168.1.67: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.67:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

RESULT:

Thus the various networks commands like tcpdump, netstat, ifconfig, nslookup and tracert, ping are executed successfully.

EX NO: 2 **HTTP web client program to download a web page using TCP sockets**

DATE:

AIM:

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

PROCEDURE:

- TCP connection with the web server using a Socket. We then use the socket's input and output streams to send an HTTP GET request and receive the server's response.
- Replace "example.com" with the actual domain name or IP address of the web page you want to download. You can also modify the port and path variables as needed.
- Keep in mind that this is a basic example, and it assumes that the web server is listening on the standard HTTP port (port 80).
- In a real-world scenario, you might need to handle redirects, handle different response codes, handle chunked encoding, and so on.

PROGRAM:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
public class WebPageDownloader
{
    public static void main(String[] args)
             {
             String host = "example.com";
             int port = 80;
            String path = "/";
             try (Socket socket = new Socket(host, port);
             PrintWriter requestWriter = new PrintWriter(socket.getOutputStream());
             BufferedReader responseReader = new BufferedReader(new
                                                            InputStreamReader(socket.getInputStream()
                                                )))
                         {

                          // Send HTTP GET request
                           requestWriter.println("GET " + path + " HTTP/1.1");
                           requestWriter.println("Host: " + host);
                           requestWriter.println("Connection: close");
                           requestWriter.println();
                           requestWriter.flush();
                         // Read and print the response
                           String line;
                           while ((line = responseReader.readLine()) != null) {
                           System.out.println(line);
```

```
                    }
                }
        catch (IOException e)
        {
                e.printStackTrace();
            }
        }
    }
```

OUT PUT:

RESULT:

Thus the above program HTTP web client program to download a web page using TCP sockets has been executed successfully

EX NO 3 (a) **Applications using TCP sockets**

DATE: **Echo client and echo server**

AIM:

Write a program for Echo Client and Echo server using TCP sockets.

PROCEDURE:

- Compile both EchoServer.java and EchoClient.java using javac.
- Start the server by running java EchoServer in one terminal or command prompt window.
- Start the client by running java EchoClient in another terminal or command prompt window.
- Type messages in the client terminal and press Enter to send them to the server.
- The server will echo the messages back to the client, and the client will display the server's response.
- Type "exit" in the client terminal to quit the program.

The Echo server listens for incoming client connections and echoes back any messages received from the client. The Echo client connects to the server and allows the user to input messages, which are then sent to the server. The server echoes the messages back to the client, which displays the server's response.

PROGRAM:

a) **Echo Server.java:**

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import  java.net.Socket;
public class EchoServer
{
   public static void main(String[] args)
        {
        int port = 12345;
        try (ServerSocket serverSocket = new ServerSocket(port))
         {
               System.out.println("EchoServer is listening on port " + port);
               while (true)
               {
```

}

```
                 Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected: " + clientSocket.getInetAddress().
                                                                              getHostAddress());
            // Start a new thread to handle the client
            Thread clientThread = new Thread(() ->
                {
                try (BufferedReader reader = new BufferedReader(new
                                    InputStreamReader(clientSocket.getInputStream()));
                 PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true))
                {
                 String line;
                while ((line = reader.readLine()) != null) {
                System.out.println("Received from client: " + line);
                writer.println("Echo: " + line);
                }
                }
                catch (IOException e)
                {
                e.printStackTrace();
                 }
                 System.out.println("Client disconnected: " +
                                          clientSocket.getInetAddress().getHostAddress());
                }
                clientThread.start();
          }
        }
          catch (IOException e)
          {
          e.printStackTrace();
          }
      }
  }
```

b) Echo Client.java:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;
public class EchoClient
 {
   public static void main(String[] args)
{
      String serverAddress = "localhost";
      int port = 12345;
            try (Socket socket = new Socket(serverAddress, port);
          BufferedReader reader = new BufferedReader(new
                                               InputStreamReader(socket.getInputStream()));
          PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
          Scanner scanner = new Scanner(System.in));
 {
          System.out.println("Connected to EchoServer at " + serverAddress + ":" + port);
         System.out.println("Type a message and press Enter to send, or type 'exit' to quit.");
 // Read input from the user and send it to the server String input;
         do
         {
            input = scanner.nextLine();
            writer.println(input);
             String response = reader.readLine();
            System.out.println("Server response: " + response);
         }
         while (!input.equalsIgnoreCase("exit"));
               }
         catch (IOException e)
         {
         e.printStackTrace();
      }
   }
}
```

Out Put:

Result:

Thus the above program for Echo Client and Echo server using TCP sockets has been executed successfully.

EX NO 3 (b)	Applications using TCP sockets
DATE:	Chat

AIM:

To write a program for the Chat Applications using TCP sockets.

PROCEDURE:

- Compile both ChatServer.java and ChatClient.java using javac.
- Start the server by running java ChatServer in one terminal or command prompt window.
- Start multiple clients by running java ChatClient in separate terminal or command prompt windows.
- Each client can type messages, and they will be broadcasted to all connected clients.

PROGRAMS:

A) ChatServer.java

```
import  java.io.*;
import java.net.*;
import java.util.*;
public class ChatServer
{
   private static final int PORT = 12345;
   private static Set<Socket> clientSockets = new HashSet<>();
   private static final Object lock = new Object();
   public static void main(String[] args)
  {
      try (ServerSocket serverSocket = new ServerSocket(PORT))
      {
         System.out.println("ChatServer is listening on port " + PORT);
         while (true)
           {
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " + clientSocket.getInetAddress().
                                                                                        getHostAddress());
            synchronized (lock)
            {
                    clientSockets.add(clientSocket);
                }
                Thread clientThread = new Thread(() -> handleClient(clientSocket));
                clientThread.start();
            }
        } catch (IOException e)
    {
            e.printStackTrace();
        }
    }

    private static void handleClient(Socket clientSocket)
    {
        try (BufferedReader reader = new BufferedReader(new InputStreamReader
                                                        (clientSocket.getInputStream()));
             PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true))
                {
            String clientMessage;
            while ((clientMessage = reader.readLine()) != null)
              {
                System.out.println("Received from client: " + clientMessage);
                // Broadcast the message to all connected clients
                synchronized (lock) {
                    for (Socket socket : clientSockets) {
                        PrintWriter clientWriter = new PrintWriter(socket.getOutputStream(), true);
                        clientWriter.println(clientMessage);
                    }
                }
            }
        }
             catch (IOException e)
              {
            e.printStackTrace();
        }
        synchronized (lock)
        {
            clientSockets.remove(clientSocket);
        }
        System.out.println("Client disconnected: " + clientSocket.getInetAddress().
                                                                getHostAddress());
    }
}
```

B) ChatClient.java

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class ChatClient
{
    private static final String SERVER_ADDRESS = "localhost";
    private static final int PORT = 12345;
    public static void main(String[] args)
{
        try (Socket socket = new Socket(SERVER_ADDRESS, PORT);
             BufferedReader reader = new BufferedReader(new
                                                InputStreamReader(socket.getInputStream()));
             PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
             Scanner scanner = new Scanner(System.in))
{
            System.out.println("Connected to ChatServer at " + SERVER_ADDRESS + ":" + PORT);
            System.out.println("Type your messages and press Enter to send, or type 'exit' to quit.");
            // Create a separate thread to receive messages from the server
            Thread receiveThread = new Thread(() ->
{
```

```
                String serverMessage;
                try
        {
                    while ((serverMessage = reader.readLine()) != null)
         {
                        System.out.println("Server says: " + serverMessage);
                    }
                } catch (IOException e)
        {
                    e.printStackTrace();
                }
            });
            receiveThread.start();
```
// Read input from the user and send messages to the server
```
            String input;
            Do
         {
                input = scanner.nextLine();
                writer.println(input);
            }
         while (!input.equalsIgnoreCase("exit"));
          }
         catch (IOException e)
        {
            e.printStackTrace();
          }
       }
}
```

OUT PUT:

RESULT:

Thus the above a program for the Chat Applications using TCP sockets has been executed succesfully.

EX NO: 4 **Simulation of DNS using UDP sockets**

DATE:

AIM:

To write a program for Simulation of DNS using UDP sockets

PROCEDURE:

The DNS server listens on port 53, which is the standard DNS port. It receives DNS queries from clients, processes them, and sends back the corresponding IP address. The process DNSQuery() method is a placeholder where you can implement your own logic to map domain names to IP addresses.

To test the DNS server, you can use a DNS client program or a tool like nslookup. Here's an example using the nslookup command:

- Compile and run the DNS server program using javac DNSServer.java and java DNSServer.
- Open a new terminal or command prompt window.
- Type nslookup example.com localhost to query the DNS server for the IP address of example.com. Replace localhost with the IP address or hostname of the machine running the DNS server.
- You should see the DNS server responding with the IP address for example.com.

Note that this is a basic example and does not cover all the complexities of a full-fledged DNS server. It's intended to demonstrate the concept of simulating DNS using UDP sockets.

PROGRAM:

```
import java.net.*;

public class DNSServer

 {

    private static final int PORT = 53;

    public static void main(String[] args)

{

        try (DatagramSocket socket = new DatagramSocket(PORT))

{

            System.out.println("DNS Server is listening on port " + PORT);

            byte[] receiveBuffer = new byte[1024];

            while (true)

 {
```

```
                DatagramPacket requestPacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
                socket.receive(requestPacket);
                InetAddress clientAddress = requestPacket.getAddress();
                int clientPort = requestPacket.getPort();
                String query = new String(requestPacket.getData()).trim();
                System.out.println("Received DNS query from client: " + query);
                // Process the DNS query and obtain the corresponding IP address
                String ipAddress = processDNSQuery(query);
                byte[] responseBytes = ipAddress.getBytes();
                DatagramPacket responsePacket = new DatagramPacket(responseBytes,
                                                    responseBytes.length, clientAddress, clientPort);
                socket.send(responsePacket);
                System.out.println("Sent DNS response to client: " + ipAddress);
            }
        }
            catch (Exception e)
            {
            e.printStackTrace();
        }
    }
    private static String processDNSQuery(String query)
  {
        // Perform DNS query processing and return the appropriate IP address based on the query
        // For simplicity, we'll use a hardcoded mapping here
        switch (query)
  {
            case "example.com":
                return "192.0.2.1";
            case "google.com":
                return "8.8.8.8";
            default:
                return "Unknown host";
        }
    }
}
```

OUT PUT:

RESULT:

Thus the above program for Simulation of DNS using UDP sockets has been executed successfully.

EX NO: 5 **Capturing Packets using WireShark**

DATE:

AIM:

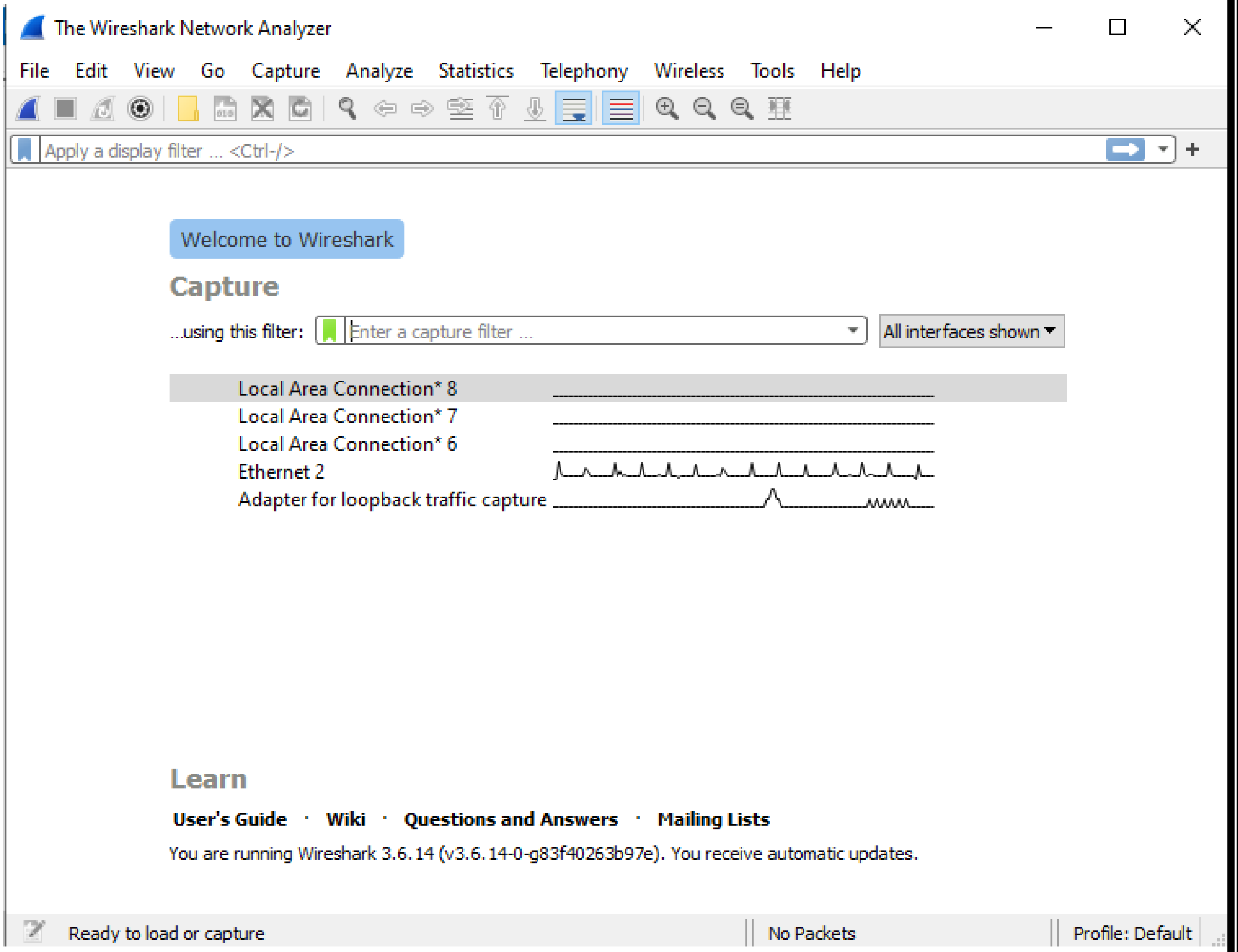
Use a tool wire shark to capture a packets and examine the packets.

PROCEDURE:

Wireshark is a powerful network protocol analyzer that allows you to capture and analyze network packets. Here's how you can use Wireshark to capture packets and examine them:

1. Start Wireshark on your machine.
2. Select the network interface that you want to capture packets from. For example, if you're using Wi-Fi, select the Wi-Fi interface. If you're using Ethernet, select the Ethernet interface.
3. Click on the "Start" button to begin capturing packets.
4. Run your DNS server program (as described in the previous response) or any other network program that generates network traffic.
5. Perform the actions or execute the program that generates the network traffic you want to capture and examine. For example, if you're using the DNS server program, you can use the nslookup command or any other DNS client program to perform DNS queries.
6. Once you're done generating the network traffic, go back to Wireshark and click on the "Stop" button to stop capturing packets.
7. Wireshark will display a list of captured packets in its main window. You can scroll through the list to view individual packets.
8. You can select a packet from the list to examine its details in the packet details pane. Wireshark provides detailed information about each packet, including source and destination IP addresses, port numbers, protocols, and packet payload.
9. To filter the captured packets based on specific criteria, you can use the filter box at the top of the Wireshark window. For example, you can filter packets by protocol, source or destination IP address, port number, or any other field.
10. You can right-click on a packet in the list to perform various actions, such as following a TCP stream, analyzing the packet in more detail, or exporting the packet data for further analysis.

Wireshark provides a wide range of features and capabilities for packet analysis, including the ability to dissect and decode various protocols, perform statistical analysis, apply filters, and much more. It's a valuable tool for examining network packets and understanding the communication between different network entities.


The Wireshark Network Analyzer

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Welcome to Wireshark

Capture

...using this filter: Enter a capture filter ... All interfaces shown

Local Area Connection* 8
Local Area Connection* 7
Local Area Connection* 6
Ethernet 2
Adapter for loopback traffic capture

Learn

User's Guide · Wiki · Questions and Answers · Mailing Lists

You are running Wireshark 3.6.14 (v3.6.14-0-g83f40263b97e). You receive automatic updates.

Ready to load or capture | No Packets | Profile: Default

RESULT:

Thus the tool wire shark to capture a packets and examine the packets has been executed successfully.

EX NO: 6 **Simulating ARP/RARP protocol**

DATE:

AIM:

To write a program to simulate ARP/RARP protocol.

PROCEDURE:

In this simulation, we have a simple ARPTable class that stores IP-MAC address mappings. The ARPProtocolSimulator class demonstrates the usage of this table to simulate ARP and RARP lookups.

To run the simulation, simply compile and execute the ARPProtocolSimulator.java file. It will perform an ARP lookup by querying the ARP table for a given IP address and display the corresponding MAC address if found. It will also perform a RARP lookup by searching the ARP table for a given MAC address and display the corresponding IP address if found.

Note that this is a basic simulation for educational purposes and doesn't involve actual network communication. In real-world scenarios, ARP and RARP involve communication between devices on a network to resolve IP and MAC addresses.

In this simulation, we have a simple RARPTable class that stores MAC-IP address mappings. The RARPProtocolSimulator class demonstrates the usage of this table to simulate a RARP lookup.

To run the simulation, simply compile and execute the RARPProtocolSimulator.java file. It will perform a RARP lookup by querying the RARP table for a given MAC address and display the corresponding IP address if found.

PROGRAM:

a) **ARP PROTOCOL**

```
import java.util.HashMap;
import  java.util.Map;
class ARPTable
{
   private Map<String, String> table;
   public ARPTable()
{
      this.table = new HashMap<>();
   }
   public void addEntry(String ipAddress, String macAddress) {
      table.put(ipAddress, macAddress);
   }
   public String lookup(String ipAddress)
{
        return table.get(ipAddress);
    }
}
class ARPProtocolSimulator
 {
    public static void main(String[] args)
{
        ARPTable arpTable = new ARPTable();
        arpTable.addEntry("192.168.0.1", "00:11:22:33:44:55");
        arpTable.addEntry("192.168.0.2", "AA:BB:CC:DD:EE:FF");
        // Simulate ARP lookup
        String ipAddress = "192.168.0.1";
        String macAddress = arpTable.lookup(ipAddress);
        if (macAddress != null)
{
            System.out.println("ARP Lookup - IP: " + ipAddress + ", MAC: " + macAddress);
        }
else
 {
            System.out.println("ARP Lookup - IP: " + ipAddress + ", MAC not found");
        }
        // Simulate RARP lookup
        String targetMacAddress = "00:11:22:33:44:55";
        String targetIpAddress = null;
        for (Map.Entry<String, String> entry : arpTable.getTable().entrySet())
{
            if (entry.getValue().equals(targetMacAddress))
{
                targetIpAddress = entry.getKey();
                break;
            }
        }
        if (targetIpAddress != null)
 {
            System.out.println("RARP Lookup - MAC: " + targetMacAddress + ", IP: " +
                                                                   targetIpAddress);
        }
 else
{
        System.out.println("RARP Lookup - MAC: " + targetMacAddress + ", IP not found");
    }
  }}
```

b) RARP PROTOCOL:

```
import java.util.HashMap;
import  java.util.Map;
class RARPTable
{
  private Map<String, String> table;
  public RARPTable()
{
    this.table = new HashMap<>();
  }
  public void addEntry(String macAddress, String ipAddress)
 {
    table.put(macAddress, ipAddress);
  }
  public String lookup(String macAddress)
 {
    return table.get(macAddress);
  }
}
class RARPProtocolSimulator
{
  public static void main(String[] args)
 {
    RARPTable rarpTable = new RARPTable();
    rarpTable.addEntry("00:11:22:33:44:55", "192.168.0.1");
    rarpTable.addEntry("AA:BB:CC:DD:EE:FF", "192.168.0.2");
    // Simulate RARP lookup
    String macAddress = "00:11:22:33:44:55";
    String ipAddress = rarpTable.lookup(macAddress);
    if (ipAddress != null)
{
        System.out.println("RARP Lookup - MAC: " + macAddress + ", IP: " + ipAddress);
```

```
                    }
            else
             {
                        System.out.println("RARP Lookup - MAC: " + macAddress + ", IP not found");
                    }
                }
            }
```

OUT PUT:

RESULT:

Thus the above program to simulate ARP/RARP protocol has been executed successfully.

EX NO: 7 **Study of Network simulator (NS)**

DATE:

AIM:

Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.

PROCEDURE:

Network Simulator (NS) is a discrete event network simulation tool used for simulating and evaluating the performance of computer networks. It allows researchers and network engineers to simulate various network scenarios, protocols, and algorithms to understand their behaviour and performance under different conditions.

NS is widely used for academic and research purposes to explore network concepts, test new protocols, study network behaviour, and evaluate network performance. It provides a flexible and extensible environment for designing network topologies, defining network parameters, and running simulations.

Some key features of NS include:

- Network Topology: NS allows users to create complex network topologies with nodes, links, and routers. It supports various network types, including wired and wireless networks.
- Protocol Simulation: NS provides a library of network protocols that can be simulated, such as TCP/IP, UDP, routing protocols (e.g., OSPF, BGP), and congestion control algorithms.
- Traffic Generation: NS allows users to generate traffic patterns and flows to simulate realistic network traffic scenarios. This enables the evaluation of network performance and congestion control algorithms under different traffic conditions.
- Event-driven Simulation: NS operates on an event-driven model, where events (such as packet arrivals, link failures, or timer expirations) are scheduled and processed based on simulation time. This allows for fine-grained control and synchronization of network events.
- Performance Metrics: NS provides built-in tools for collecting and analysing performance metrics during simulations. These metrics can include throughput, latency, packet loss, congestion indicators, and more.

Now, let's discuss simulating Congestion Control Algorithms using NS. Congestion control algorithms aim to manage network congestion and optimize network performance by regulating the rate at which data is transmitted. NS enables researchers to simulate and evaluate different congestion control algorithms and compare their performance in various network scenarios.

To simulate congestion control algorithms using NS, you would typically follow these steps:

Design the network topology:

Define the network topology you want to simulate, including nodes, links, and their properties.

Implement the congestion control algorithm:

Write the code or configure the congestion control algorithm within NS. NS provides a programming interface to implement custom congestion control algorithms or use existing ones available in the library.

Define traffic patterns:

Specify the traffic patterns and flows that will be generated in the network. This can include different types of traffic, such as bulk data transfers, real-time traffic, or background traffic.

Run the simulation: Start the simulation and let NS simulate the network behaviour over time. NS will handle events, process packets, and simulate the congestion control algorithm's actions based on the defined network conditions and traffic patterns.

Collect and analyse results:

Monitor and collect performance metrics during the simulation, such as throughput, delay, packet loss, and congestion indicators. Analyse the results to evaluate the effectiveness and efficiency of the congestion control algorithm under different scenarios.

Iterate and refine:

Based on the simulation results, refine the congestion control algorithm or experiment with different parameter settings to optimize network performance.

NS provides extensive documentation, tutorials, and examples to help users get started with simulating congestion control algorithms and exploring network behaviour. You can refer to the official NS documentation and resources for detailed guidance on using NS for congestion control simulations.

Keep in mind that simulating congestion control algorithms in NS provides a controlled environment for evaluation and comparison. Real-world network deployments may introduce additional complexities and factors that can impact the algorithm's performance. Therefore, it's essential to validate the simulation results with real-world experiments when possible.

RESULT:

Thus the above study of Network Stimulator has been studied and understood properly.

EX NO: 8 **Study of TCP/UDP using simulation tool**

DATE:

AIM:

Study of TCP/UDP using simulation tool.

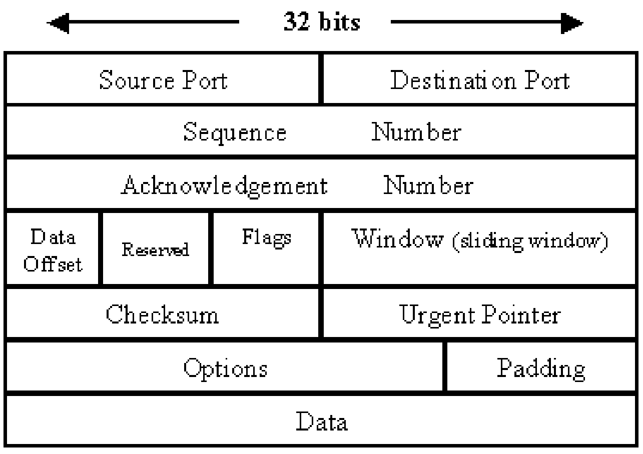
PROCEDURE:

To study the performance of TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) using simulation tools, you can use network simulators like Network Simulator (NS), ns-3, or OMNeT++ that provide support for simulating these protocols. Here's an overview of the process:

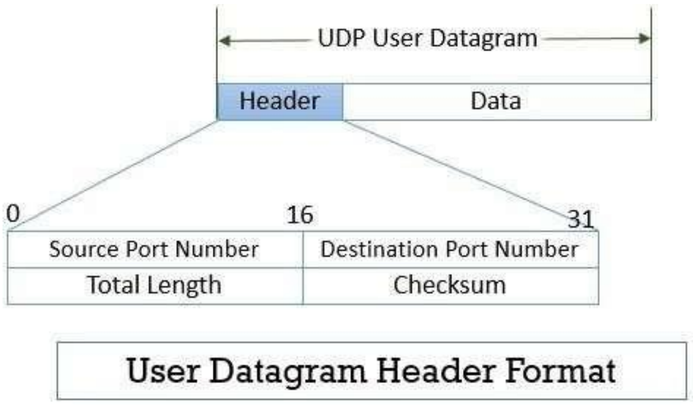
Network Topology Design:

Start by designing the network topology that you want to simulate. Determine the number of nodes, their interconnections, and the characteristics of the links (e.g., bandwidth, delay, loss rate). Consider the network scenario that best represents your study, such as a local area network (LAN) or wide area network (WAN).

TCP Header Format:

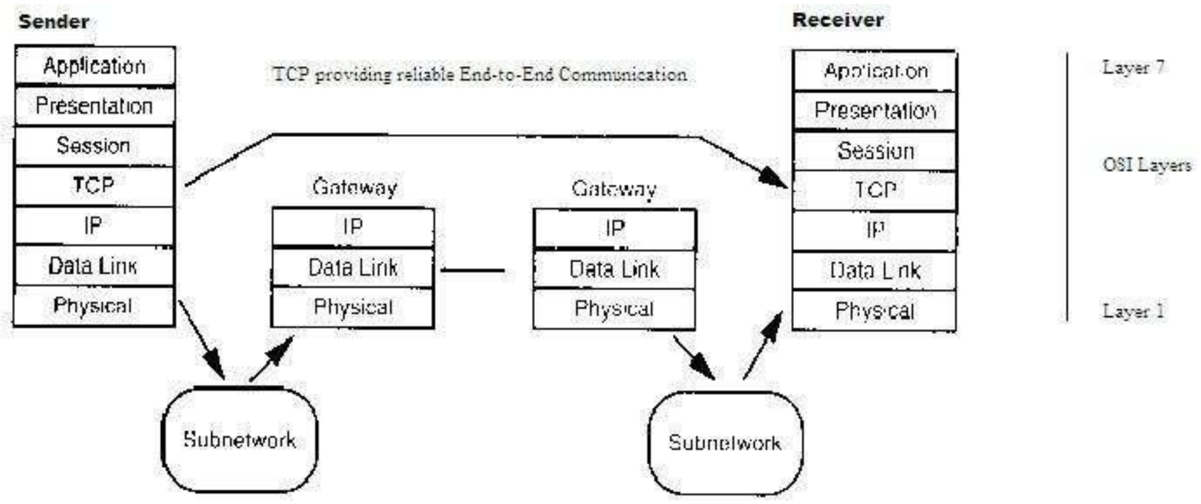


UDP Header Format:



Protocol Configuration:

Configure the simulation environment to use TCP and UDP protocols. Specify the parameters and settings related to these protocols, such as TCP congestion control algorithms (e.g., Reno, Cubic) or UDP packet sizes.



Traffic Generation:

Define the traffic patterns and flows that will be generated in the network. Decide on the types of traffic, such as bulk transfers, video streaming, or real-time communications. Set the traffic characteristics, such as arrival rates, sizes, and destinations, to mimic real-world scenarios.

Simulation Execution:

Run the simulation using the chosen simulation tool. During the simulation, the tool will model the behaviour of TCP and UDP based on the configured parameters and network conditions. Packets will be sent, routed, and received by the simulated nodes, and the protocols' mechanisms will be employed, such as TCP congestion control or UDP's best-effort delivery.

Performance Metrics Collection:

Monitor and collect performance metrics during the simulation to evaluate the TCP and UDP performance. Common metrics include throughput, latency, packet loss rate, end-to-end delay, fairness, and congestion indicators. The simulation tool should provide mechanisms to gather these metrics for analysis.

Analysis and Comparison:

Analyse the collected data to assess the performance of TCP and UDP. Compare their behavior under different scenarios, varying network conditions, or congestion levels. Evaluate the impact of protocol settings, such as window size or timeout values, on the performance metrics. Identify strengths, weaknesses, and trade-offs between TCP and UDP in the simulated environment.

Validation and Verification:

Validate the simulation results by comparing them with theoretical expectations or known behaviours of TCP and UDP. If possible, validate the results against real-world measurements or experiments to ensure the simulation accurately represents the protocol performance.

Simulation tools like NS, ns-3, or OMNeT++ provide extensive documentation, example scenarios, and libraries that facilitate the simulation of TCP and UDP. You can consult their documentation and resources to learn more about using these tools to study the performance of TCP and UDP protocols in different network scenarios.

Keep in mind that while simulations provide valuable insights, they may not fully capture the complexity of real-world networks. Therefore, it's important to complement the simulation results with real-world experiments whenever possible to validate and verify the findings.

RESULT:

Thus the above study of TCP/UDP protocol has been studied and understood properly.

EX NO: 9(a) **Simulation of Distance Vector Algorithm**

DATE:

AIM:

To simulate the Distance Vector routing algorithm

PROCEDURE:

To simulate the Distance Vector routing algorithm, you can use network simulation tools like Network Simulator (NS), ns-3, or OMNeT++. Here's a step-by-step guide on simulating the Distance Vector algorithm:

Network Topology Design:

Design the network topology you want to simulate. Define the nodes and links in the network, including their properties such as bandwidth, delay, and reliability. Consider the size and complexity of the network to reflect the scenario you want to study.

Node Configuration:

Configure each node in the network to use the Distance Vector routing algorithm. Each node should maintain its routing table and exchange routing updates with its neighboring nodes.

Routing Table Initialization:

Initialize the routing tables of each node in the network. Set the initial costs to neighboring nodes (directly connected links) and infinity for all other nodes.

Routing Update Exchange:

Implement the routing update exchange process. Each node should periodically send its routing table to its neighboring nodes and receive routing updates from them. The updates contain information about the costs to reach different destinations.

Distance Vector Calculation:

Implement the distance vector calculation mechanism. Each node should calculate the shortest path to each destination based on the received routing updates and update its routing table accordingly.

Routing Table Updates:

Whenever a node's routing table is updated, it should inform its neighboring nodes about the changes by sending routing updates.

Simulation Execution:

Run the simulation using the selected simulation tool. The simulation will execute the Distance Vector algorithm, update the routing tables, and forward packets based on the calculated paths.

Traffic Generation:

Define the traffic patterns and flows to be used in the simulation. Generate traffic between nodes to simulate realistic network conditions. Specify the type, volume, and source-destination pairs of the traffic flows based on your study objectives.

Performance Metrics Collection:

Monitor and collect performance metrics during the simulation to evaluate the Distance Vector algorithm's performance. Common metrics include routing convergence time, packet delivery ratio, end-to-end delay, routing table size, and network utilization. The simulation tool should provide mechanisms to collect these metrics for analysis.

Analysis and Comparison:

Analyze the collected data to assess the performance of the Distance Vector algorithm. Compare its behavior under different scenarios, network sizes, and traffic patterns. Evaluate its convergence time, routing stability, scalability, and ability to handle network changes.

Validation and Verification:

Validate the simulation results by comparing them with theoretical expectations or known behaviors of the Distance Vector algorithm. If possible, validate the results against real-world measurements or experiments to ensure the simulation accurately represents the algorithm's performance.

Simulation tools like NS, ns-3, or OMNeT++ provide libraries, modules, and example scenarios to facilitate the simulation of the Distance Vector algorithm. You can refer to their documentation and resources for specific guidance on using these tools for Distance Vector simulations.

Remember that simulations provide insights into the behavior of routing algorithms but may not fully capture the complexity of real-world networks. Therefore, it's important to complement simulation results with real-world experiments whenever possible to validate and verify the findings.

RESULT:

Thus the above simulation program has been executed successfully.

EX NO: 9(b) **Simulation of Link State Routing algorithm**

DATE:

AIM:

Simulation of Link State Routing algorithm.

PROCEDURE:

To simulate the Link State routing algorithm, you can use network simulation tools like Network Simulator (NS), ns-3, or OMNeT++. Here's a step-by-step guide on simulating the Link State algorithm:

Network Topology Design:

Design the network topology you want to simulate. Define the nodes and links in the network, including their properties such as bandwidth, delay, and reliability. Consider the size and complexity of the network to reflect the scenario you want to study.

Node Configuration:

Configure each node in the network to use the Link State routing algorithm. Each node should have a Link State Database (LSDB) that stores information about the network's topology.

Link State Information Exchange:

Implement the link state information exchange process. Each node should flood its link state information to all other nodes in the network. This involves creating Link State Packets (LSPs) that contain information about the node's links and flooding them throughout the network.

Link State Database Update:

When a node receives an LSP from another node, it should update its LSDB with the received information. Each node's LSDB should represent the complete network topology based on the received LSPs.

Shortest Path Calculation:

Implement the shortest path calculation mechanism. Each node should run a shortest path algorithm, such as Dijkstra's algorithm, using the information in its LSDB to determine the shortest paths to each destination node.

Routing Table Construction:

Based on the shortest paths calculated, each node should construct its routing table. The routing table should specify the next hop for each destination node.

Packet Forwarding:

During the simulation, when a packet needs to be forwarded, the source node consults its routing table to determine the next hop and forwards the packet accordingly.

Simulation Execution:

Run the simulation using the selected simulation tool. The simulation will execute the Link State algorithm, update the LSDBs, calculate shortest paths, and forward packets based on the routing tables.

Traffic Generation:

Define the traffic patterns and flows to be used in the simulation. Generate traffic between nodes to simulate realistic network conditions. Specify the type, volume, and source-destination pairs of the traffic flows based on your study objectives.

Performance Metrics Collection:

Monitor and collect performance metrics during the simulation to evaluate the Link State algorithm's performance. Common metrics include routing convergence time, packet delivery ratio, end-to-end delay, routing table size, and network utilization. The simulation tool should provide mechanisms to collect these metrics for analysis.

Analysis and Comparison:

Analyze the collected data to assess the performance of the Link State algorithm. Compare its behavior under different scenarios, network sizes, and traffic patterns. Evaluate its convergence time, routing stability, scalability, and ability to handle network changes.

Validation and Verification:

Validate the simulation results by comparing them with theoretical expectations or known behaviors of the Link State algorithm. If possible, validate the results against real-world measurements or experiments to ensure the simulation accurately represents the algorithm's performance.

Simulation tools like NS, ns-3, or OMNeT++ provide libraries, modules, and example scenarios to facilitate the simulation of the Link State algorithm. You can refer to their documentation and resources for specific guidance on using these tools for Link State simulations.

Remember that simulations provide insights into the behavior of routing algorithms but may not fully capture the complexity of real-world networks. Therefore, it's important to complement simulation results with real-world experiments whenever possible to validate and verify the findings.

RESULT:

Thus the above simulation program has been executed successfully.

EX NO: 10 **Simulation of an error correction code (CRC)**

DATE:

AIM:

Simulation of an error correction code (CRC).

PROCEDURE:

To simulate an error correction code like CRC (Cyclic Redundancy Check), you can use programming languages like Python to create a simulation program. Here's a step-by-step guide on simulating CRC:

Select CRC Parameters:

Determine the parameters of the CRC code you want to simulate, such as the polynomial and its degree, generator polynomial, and the number of bits for the checksum. These parameters define the characteristics of the CRC code.

Generate Test Data:

Create test data that represents the message to be encoded and transmitted. This can be a string of bits or bytes, or you can generate random data. Make sure the data contains errors or introduce errors intentionally to simulate the error correction process.

CRC Encoding:

Implement the CRC encoding process. This involves dividing the message by the generator polynomial using modulo-2 division and appending the resulting CRC checksum to the original message. This generates the encoded message.

Error Introduction:

Introduce errors into the encoded message to simulate transmission errors. You can flip bits randomly or intentionally introduce specific error patterns.

CRC Decoding:

Implement the CRC decoding process. Receive the transmitted message and perform the same modulo-2 division with the generator polynomial. If the remainder is zero, the message is assumed to be error-free. If the remainder is non-zero, it indicates the presence of errors.

Error Detection and Correction:

Analyze the remainder obtained in the decoding process. If the remainder is non-zero, errors are detected. Use the CRC algorithm to correct the errors, if possible. You can attempt to fix the errors by manipulating the received message or requesting retransmission.

Performance Analysis:

Collect performance metrics to evaluate the effectiveness of the CRC code. Measure the error detection rate, error correction rate, and false positive rate. Calculate the overall accuracy of the CRC code in detecting and correcting errors.

Repeat and Compare:

Repeat the simulation with different test data, error patterns, or CRC parameters to analyze the performance under various conditions. Compare the results to assess the impact of different factors on the CRC's effectiveness.

By implementing the CRC algorithm in a programming language, you can simulate the error detection and correction process. You can modify the simulation program to test different CRC parameters, error patterns, or even combine CRC with other error correction techniques.

Remember that this simulation represents a simplified version of real-world scenarios, and CRC is just one type of error correction code. More complex error correction codes, such as Reed-Solomon codes, may require additional considerations and calculations.

RESULT:

Thus the above simulation program has been executed successfully.