DISCLAIMER

This PDF has been created according to my specific preferences and requirements. All users are advised to thoroughly review the document before printing and make any modifications deemed necessary based on their own judgment. I will not be held responsible for any errors, misunderstandings, or oversights on the part of others.

If you have any objections to the content or structure, please feel free to create and share an alternative version with the group. **DO NOT USE THIS if you do not agree with its contents**.

இந்த PDF எனது குறிப்பிட்ட விருப்பங்கள் மற்றும் தேவைகளுக்கு ஏற்ப உருவாக்கப்பட்டது. அனைத்து பயனர்களும் அச்சிடுவதற்கு முன் ஆவணத்தை முழுமையாக மதிப்பாய்வு செய்து, தங்கள் சொந்த தீர்ப்பின் அடிப்படையில் தேவையான மாற்றங்களைச் செய்யுமாறு அறிவுறுத்தப்படுகிறார்கள்.
மற்றவர்களின் தவறுகள், தவறான புரிதல்கள் அல்லது மேற்பார்வைகளுக்கு நான் பொறுப்பல்ல.

உள்ளடக்கம் அல்லது கட்டமைப்பில் ஏதேனும் ஆட்சேபனைகள் இருந்தால், நீங்கள் மாற்று பதிப்பை உருவாக்கி அதை குழுவுடன் பகிர்ந்து கொள்ளலாம். **அதன் உள்ளடக்கங்களுடன் நீங்கள் உடன்படவில்லை என்றால் பயன்படுத்த வேண்டாம்**.

यह पीडीएफ मेरी विशिष्ट इच्छाओं और आवश्यकताओं के अनुसार बनाया गया था। सभी उपयोगकर्ताओं को सलाह दी जाती है कि वे प्रिंट करने से पहले दस्तावेज़ की अच्छी तरह से समीक्षा करें और अपने स्वयं के निर्णय के आधार पर कोई भी आवश्यक परिवर्तन करें। मैं दूसरों की गलतियों, गलतफहमियों या चूक के लिए जिम्मेदार नहीं हूँ।

यदि आपको सामग्री या संरचना पर कोई आपत्ति है, तो आप एक वैकल्पिक संस्करण बना सकते हैं और इसे टीम के साथ साझा कर सकते हैं। **यदि आप इसकी सामग्री से सहमत नहीं हैं तो इसका उपयोग न करें।**

THANK YOU

PROGRAM:

MyClient.java

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

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

MyServer.java

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

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

PROGRAM:

client.java

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

class client {
    public static void main(String a[]) throws Exception {
        Socket s = new Socket(InetAddress.getLocalHost(), 10);
        DataInputStream in = new DataInputStream(s.getInputStream());
        PrintStream p = new PrintStream(s.getOutputStream());
        int i = 0, rptr = -1, nf, rws = 8;
        String rbuf[] = new String[8];
        String ch;
        System.out.println();
        do {
            nf = Integer.parseInt(in.readLine());
            if (nf <= rws - 1) {
                for (i = 1; i <= nf; i++) {
                    rptr = ++rptr % 8;
                    rbuf[rptr] = in.readLine();
                    System.out.println("The received Frame " + rptr + " is : " + rbuf[rptr]);
                }
                rws -= nf;
                System.out.println("\nAcknowledgment sent\n");
                p.println(rptr + 1);
                rws += nf;
            } else
                break;
            ch = in.readLine();
        } while (ch.equals("yes"));
    }
}
```

Server.java

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

public class server {
    public static void main(String a[]) throws Exception {
        ServerSocket ser = new ServerSocket(10);
        Socket s = ser.accept();
        DataInputStream in = new DataInputStream(System.in);
        DataInputStream in1 = new DataInputStream(s.getInputStream());
        String sbuff[] = new String[8];
        PrintStream p;
        int sptr = 0, sws = 8, nf, ano, i;
        String ch;
        do {
            p = new PrintStream(s.getOutputStream());
            System.out.print("Enter the no. of frames : ");
            nf = Integer.parseInt(in.readLine());
            p.println(nf);
            if (nf <= sws - 1) {
                System.out.println("Enter " + nf + " Messages to be send\n");
                for (i = 1; i <= nf; i++) {
                    sbuff[sptr] = in.readLine();
                    p.println(sbuff[sptr]);
                    sptr = ++sptr % 8;
                }
                sws -= nf;
                System.out.print("Acknowledgment received");
                ano = Integer.parseInt(in1.readLine());
                System.out.println(" for " + ano + " frames");
                sws += nf;
            } else {
                System.out.println("The no. of frames exceeds window size");
                break;
            }
            System.out.print("\nDo you wants to send some more frames : ");
            ch = in.readLine();
            p.println(ch);
        } while (ch.equals("yes"));
        s.close();
    }
}
```

PROGRAM:

(a) Program for Address Resolution Protocol (RARP) using UDP

Clientarp.java

```
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", 140);
         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);
      }
   }
}
```

Serverarp.java

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

class Serverarp {
   public static void main(String args[]) {
      try {
         ServerSocket obj = new ServerSocket(140);
         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();
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

(b) Program for Reverse Address Resolution Protocol (RARP) using UDP

Clientrarp.java

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

class Clientrarp {
   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);
      }
   }
}
```

Serverrarp.java

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

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

PROGRAM:

DaytimeClient.java

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

public class DaytimeClient {
   public static final int SERVICE_PORT = 8080;
   public static void main(String[] args) {
      if (args.length != 1) {
         System.out.println("Syntax: java DaytimeClient <hostname>");
         return;
      }
      String hostname = args[0];
      try {
         Socket daytime = new Socket(hostname, SERVICE_PORT);
         System.out.println("Connection established");
         daytime.setSoTimeout(5000);
         BufferedReader reader = new BufferedReader(
            new InputStreamReader(daytime.getInputStream(), "UTF-8")
         );
         System.out.println("Server response: " + reader.readLine());
         daytime.close();
      } catch (IOException ioe) {
         System.err.println("Error: " + ioe);
      }
   }
}
```

SimpleDaytimeServer.java

```
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.nio.charset.Charset;
import java.nio.charset.CharsetEncoder;

public class SimpleDaytimeServer {
    public static void main(String[] args) throws java.io.IOException {
        int port = 8080;
        if (args.length > 0) {
            port = Integer.parseInt(args[0]);
        }
        ServerSocketChannel server = ServerSocketChannel.open();
        server.socket().bind(new InetSocketAddress(port));
        CharsetEncoder encoder = Charset.forName("UTF-8").newEncoder();
        System.out.println("Daytime server started on port " + port);
        while (true) {
            SocketChannel client = server.accept();
            String date = new java.util.Date().toString() + "\r\n";
            ByteBuffer response = encoder.encode(CharBuffer.wrap(date));
            client.write(response);
            client.close();
        }
    }
}
```

PROGRAM:

EClient.java

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

public class EClient {
    public static void main(String arg[]) {
        Socket c = null;
        String line;
        BufferedReader is, is1;
        PrintStream os;
        try {
            InetAddress ia = InetAddress.getLocalHost();
            c = new Socket(ia, 9000);
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            if (c != null) {
                System.out.println("Connected to the server.");

                os = new PrintStream(c.getOutputStream());
                is = new BufferedReader(new InputStreamReader(System.in));
                is1 = new BufferedReader(new InputStreamReader(c.getInputStream()));
                while (true) {
                    System.out.print("Client: ");
                    line = is.readLine();
                    os.println(line);
                    System.out.println("Server: " + is1.readLine());
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (c != null) c.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

EServer.java

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

public class EServer {
    public static void main(String args[]) {
        ServerSocket s = null;
        Socket c = null;
        String line;
        BufferedReader is;
        PrintStream ps;
        try {
            s = new ServerSocket(9000);
            System.out.println("Server started. Waiting for a client...");
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            if (s != null) {
                c = s.accept();
                System.out.println("Client connected.");
                is = new BufferedReader(new InputStreamReader(c.getInputStream()));
                ps = new PrintStream(c.getOutputStream());
                while (true) {
                    line = is.readLine();
                    ps.println(line);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (c != null) c.close();
                if (s != null) s.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

PROGRAM:

UDPclient.java

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

class UDPclient {
   public static DatagramSocket ds;
   public static int clientport = 789, serverport = 790;

   public static void main(String args[]) throws Exception {
      byte buffer[] = new byte[1024];
      ds = new DatagramSocket(serverport);
      BufferedReader dis = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("server waiting");
      InetAddress ia = InetAddress.getLocalHost();
      while (true) {
         System.out.print("Client:");
         String str = dis.readLine();
         if (str.equals("end"))
            break;
         buffer = str.getBytes();
         ds.send(new DatagramPacket(buffer, str.length(), ia, clientport));
         DatagramPacket p = new DatagramPacket(buffer, buffer.length);
         ds.receive(p);
         String psx = new String(p.getData(), 0, p.getLength());
         System.out.println("Server:" + psx);
      }
   }
}
```

UDPserver.java

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

class UDPserver {
    public static DatagramSocket ds;
    public static byte buffer[] = new byte[1024];
    public static int clientport = 789, serverport = 790;

    public static void main(String args[]) throws Exception {
        ds = new DatagramSocket(clientport);
        System.out.println("press ctrl+c to quit the program");
        BufferedReader dis = new BufferedReader(new InputStreamReader(System.in));
        InetAddress ia = InetAddress.getLocalHost();
        while (true) {
            DatagramPacket p = new DatagramPacket(buffer, buffer.length);
            ds.receive(p);
            String psx = new String(p.getData(), 0, p.getLength());
            System.out.println("Client:" + psx);
            System.out.print("Server:");
            String str = dis.readLine();
            if (str.equals("end"))
                break;
            buffer = str.getBytes();
            ds.send(new DatagramPacket(buffer, str.length(), ia, serverport));
        }
    }
}
```

PROGRAM:

FileServer.java

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

public class FileServer {
    public static void main(String[] args) throws Exception {
        ServerSocket ssock = new ServerSocket(5000);
        Socket socket = ssock.accept();
        InetAddress IA = InetAddress.getByName("localhost");
        File file = new File("C:\\Users\\17_karthick_03\\Downloads\\CN Lab\\index.html");
        FileInputStream fis = new FileInputStream(file);
        BufferedInputStream bis = new BufferedInputStream(fis);
        OutputStream os = socket.getOutputStream();
        byte[] contents;
        long fileLength = file.length();
        long current = 0;
        long start = System.nanoTime();
        while (current != fileLength) {
            int size = 10000;
            if (fileLength - current >= size)
                current += size;
            else {
                size = (int) (fileLength - current);
                current = fileLength;
            }
            contents = new byte[size];
            bis.read(contents, 0, size);
            os.write(contents);
            System.out.println("Sending file ... " + (current * 100) / fileLength + "% complete!");
        }
        os.flush();
        socket.close();
        ssock.close();
        System.out.println("File sent succesfully!");
    }
}
```

FileClient.java

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.Socket;

public class FileClient {
   public static void main(String[] args) throws Exception {
      Socket socket = new Socket(InetAddress.getByName("localhost"), 5000);
      byte[] contents = new byte[10000];
      FileOutputStream fos = new
      FileOutputStream("C:\\Users\\17_karthick_03\\Downloads\\CN Lab\\index_1.html");
      BufferedOutputStream bos = new BufferedOutputStream(fos);
      InputStream is = socket.getInputStream();
      int bytesRead = 0;
      while ((bytesRead = is.read(contents)) != -1)
         bos.write(contents, 0, bytesRead);
      bos.flush();
      socket.close();
      System.out.println("File saved successfully!");
   }
}
```

PROGRAM:

Server_7.java

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

class Server_7 {
   @SuppressWarnings("resource")
   public static void main(String args[]) throws Exception {
      ServerSocket server = null;
      Socket socket;
      server = new ServerSocket(4000);
      System.out.println("Server Waiting for image");
      socket = server.accept();
      System.out.println("Client connected.");
      InputStream in = socket.getInputStream();
      DataInputStream dis = new DataInputStream(in);
      int len = dis.readInt();
      System.out.println("Image Size: " + len / 1024 + "KB");
      byte[] data = new byte[len];
      dis.readFully(data);
      dis.close();
      in.close();
      InputStream ian = new ByteArrayInputStream(data);
      BufferedImage bImage = ImageIO.read(ian);
      JFrame f = new JFrame("Server");
      ImageIcon icon = new ImageIcon(bImage);
      JLabel l = new JLabel();
      l.setIcon(icon);
      f.add(l);
      f.pack();
      f.setVisible(true);
   }
}
```

Client_7.java

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

public class Client_7 {
   public static void main(String args[]) throws Exception {

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

PROGRAM:

udpdnsserver.java

```
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 = { "trackmyclass.site", "yahoo.com", "gmail.com", "cricinfo.com", 
"facebook.com" };
      String[] ip = { "185.199.108.153","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();
      }
   }
}
```

udpdnsclient.java

```
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();
      InetAddress ipaddress;
      if (args.length == 0)
         ipaddress = InetAddress.getLocalHost();
      else
         ipaddress = InetAddress.getByName(args[0]);
      byte[] senddata = new byte[1024];
      byte[] receivedata = new byte[1024];
      int portaddr = 1362;
      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);
      clientsocket.close();
   }
}
```

PROGRAM:

Link State Routing Protocol (linkstate.tcl)

```
set ns [new Simulator]
$ns rtproto LS
set nf [open linkstate.nam w]
$ns namtrace-all $nf
set f0 [open linkstate.tr w]
$ns trace-all $f0
proc finish {} {
   global ns f0 nf
   $ns flush-trace
   close $f0
   close $nf
   exec nam linkstate.nam &
   exit 0
}
for {set i 0} {$i < 7} {incr i} {
   set n($i) [$ns node]
}
for {set i 0} {$i < 7} {incr i} {
   $ns duplex-link $n($i) $n([expr ($i+1) % 7]) 1Mb 10ms DropTail
}
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(3) $null0
$ns connect $udp0 $null0
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

PROGRAM:

Distance Vector Routing Algorithm (distvect.tcl)

```
# Distance Vector Routing Protocol – distvect.tcl
# Create a simulator object
set ns [new Simulator]
# Use distance vector routing
$ns rtproto DV
# Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
# Open the trace file
set nt [open trace.tr w]
$ns trace-all $nt
# Define the 'finish' procedure
proc finish {} {
   global ns nf nt
   $ns flush-trace
   close $nf
   close $nt
   exec nam out.nam &
   exit 0
}
# Create 8 nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
# Specify link characteristics
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns duplex-link $n5 $n6 1Mb 10ms DropTail
$ns duplex-link $n6 $n7 1Mb 10ms DropTail
$ns duplex-link $n7 $n8 1Mb 10ms DropTail
$ns duplex-link $n8 $n1 1Mb 10ms DropTail
# Specify layout as an octagon
$ns duplex-link-op $n1 $n2 orient left-up
$ns duplex-link-op $n2 $n3 orient up
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n4 $n5 orient right
```

```
$ns duplex-link-op $n5 $n6 orient right-down
$ns duplex-link-op $n6 $n7 orient down
$ns duplex-link-op $n7 $n8 orient left-down
$ns duplex-link-op $n8 $n1 orient left
# Create a UDP agent and attach it to node n1
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
# Create a Null agent (a traffic sink) and attach it to node n4
set null0 [new Agent/Null]
$ns attach-agent $n4 $null0
# Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
# Schedule events for the CBR agent and the network dynamics
$ns at 0.0 "$n1 label Source"
$ns at 0.0 "$n4 label Destination"
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n3 $n4
$ns rtmodel-at 2.0 up $n3 $n4
$ns at 4.5 "$cbr0 stop"
# Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
# Run the simulation
$ns run
```

PROGRAM:

TCP using NS2 (simulation.tcl)

```
set ns [new Simulator]
$ns color 0 Blue
$ns color 1 Red
$ns color 2 Yellow
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set f [open tcpout.tr w]
$ns trace-all $f
set nf [open tcpout.nam w]
$ns namtrace-all $nf
$ns duplex-link $n0 $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n2 $n3 queuePos 0.5
set tcp [new Agent/TCP]
$tcp set class_ 1
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $tcp
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.2 "$ftp start"
$ns at 1.35 "$ns detach-agent $n1 $tcp ; $ns detach-agent $n3 $sink"
$ns at 3.0 "finish"
proc finish {} {
   global ns f nf
   $ns flush-trace
   close $f
   close $nf
   puts "Running nam.."
   exec xgraph tcpout.tr -geometry 600x800 &
   exec nam tcpout.nam &
   exit 0
}
$ns run
```

PROGRAM:

UDP using NS2 (udp_simulation.tcl)

```
set ns [new Simulator]
$ns color 0 Blue
$ns color 1 Red
$ns color 2 Yellow
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set f [open udpout.tr w]
$ns trace-all $f
set nf [open udpout.nam w]
$ns namtrace-all $nf
$ns duplex-link $n0 $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n2 $n3 queuePos 0.5
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set udp1 [new Agent/UDP]
$ns attach-agent $n3 $udp1
$udp1 set class_ 0
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
set null1 [new Agent/Null]
$ns attach-agent $n1 $null1
$ns connect $udp0 $null0
$ns connect $udp1 $null1
$ns at 1.0 "$cbr0 start"
$ns at 1.1 "$cbr1 start"
puts [$cbr0 set packetSize_]
puts [$cbr0 set interval_]
$ns at 3.0 "finish"
proc finish {} {
   global ns f nf
   $ns flush-trace
   close $f
```

```
    close $nf
    puts "Running nam.."
    exec nam udpout.nam &
    exit 0
}
$ns run
```