PART - A

# Program 1. Write a networking program in Java to implement a TCP server that provides services for a TCP Client.

**TCP Client -**

|  |
| --- |
| import java.io.\*;  import java.net.\*; |
| class TCPClient{ |
| public static void main(String[] args){ |
| Socket client; |
| InputStream inputStream; |
| DataInputStream dataInputStream; |
| try{ |
| client = new Socket("localhost", 7313); |
| inputStream = client.getInputStream(); |
| dataInputStream =new DataInputStream(inputStream); |
| System.out.println(dataInputStream.readUTF()); |
| System.out.println(dataInputStream.readUTF()); |
| client.close(); |
| }catch(IOException e){ |
| System.out.println(e); |
| } |
| } |
| } |

**TCP Server -**

import java.io.\*; import java.net.\*;

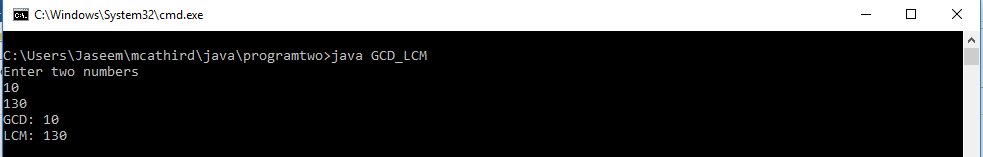
|  |
| --- |
| import java.util.\*; |
| class TCPServer{ |
| public static void main(String[] args){ |
| ServerSocket server; |
| Socket client; |
| OutputStream outputStream; |
| DataOutputStream dataOutputStream; |
| Calendar calendar; |
| try{ |
| server = new ServerSocket(7313); |
| System.out.println("Server started..."); |
| client = server.accept(); |
| System.out.println("Connected:" + client.getInetAddress()); |
| outputStream = client.getOutputStream(); |
| dataOutputStream = new DataOutputStream(outputStream); |
| dataOutputStream.writeUTF("Hi from server. We provide time service."); |
| calendar = Calendar.getInstance(); |
| dataOutputStream.writeUTF("Time:"+ calendar.get(Calendar.HOUR\_OF\_DAY) + ":" +  calendar.get(Calendar.MINUTE) |
| + ":" + calendar.get(Calendar.SECOND)); |
| server.close(); |
| }catch(IOException e){ |
| System.out.println(e); |
| } |
| } |
| } |

Output:

Output:

# Program 2. Write a networking program to implement socket programming using User datagram Protocol in Java.

**UDP Client –**



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

|  |
| --- |
| public class UDPClient { |
| public static void main(String[] args){ |
| DatagramSocket datagramSocket; |
| DatagramPacket datagramPacket; |
| String userInput; |
| InetAddress ipAddr; |
| Scanner scanner = new Scanner(System.in); |
| byte[] bytes = new byte[1024]; |
| try{ |
| datagramSocket = new DatagramSocket(7314); |
| ipAddr = InetAddress.getByName("localhost"); |
| System.out.println("Write msg to send"); |
| userInput = scanner.nextLine(); |
| bytes = userInput.getBytes(); |
| datagramPacket =new DatagramPacket(bytes,bytes.length,ipAddr,7313); |
| datagramSocket.send(datagramPacket); |
| scanner.close(); |
| } catch (SocketException ex) { |
| System.out.println(ex); |
| } catch (UnknownHostException ex) { |
| System.out.println(ex); |
| } catch (IOException ex) { |
| System.out.println(ex); |
| } |
| } |
| } |

**UDP Server –**

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

|  |
| --- |
| public static void main(String[] args) { |
| DatagramSocket datagramSocket; |
| DatagramPacket datagramPacket; |
| byte[] bytes; |
| String msg; |
| try { |
| datagramSocket = new DatagramSocket(7313); |
| System.out.println("UDP Server started..."); |
| bytes = new byte[1024]; |
| datagramPacket = new DatagramPacket(bytes, 0, bytes.length); |
| datagramSocket.receive(datagramPacket); |
| msg = new String(bytes); |
| System.out.println(msg); |
| } catch (SocketException ex) { |
| System.out.println(ex); |
| } catch (IOException ex) { |
| System.out.println(ex); |
| } |
| } |
| } |

Output:

# Program 3. Implement an FTP server using socket programming.

**FTP Client -**

import java.net.\*; import java.io.\*;

class FTPClient{

public static void main(String[] args){

String fileName; File file;

FileOutputStream fileOutputStream = null;

InputStream inputStream = null; DataInputStream dataInputStream = null; Socket socket = null;

byte[] byteArray; try {

socket = new Socket("localhost", 7313); System.out.println("Connected...");

inputStream = socket.getInputStream(); dataInputStream = new DataInputStream(inputStream); fileName = dataInputStream.readUTF();

file = new File(fileName);

fileOutputStream = new FileOutputStream(file);

Output:

int bytesRead = 0;

byteArray = new byte[1024 \* 10];

while((bytesRead = inputStream.read(byteArray)) > -1){ fileOutputStream.write(byteArray, 0, bytesRead);

}

System.out.println("Received Successfully..."); fileOutputStream.close();

socket.close();

} catch (IOException e) { System.err.println("Error 1"); e.printStackTrace();

}

}

}

**FTP Server -**

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

class FTPServer{

public static void main(String[] args){ String fileName = null; FileInputStream fileInputStream = null; OutputStream outputStream = null;

DataOutputStream dataOutputStream = null; ServerSocket serverSocket = null;

Socket client = null;

Scanner scan = new Scanner(System.in); File file;

byte[] bytes; try

{

serverSocket = new ServerSocket(7313); System.out.println("FTP Server started..."); client = serverSocket.accept(); System.out.println("Client connected..."); outputStream = client.getOutputStream();

dataOutputStream = new DataOutputStream(outputStream); System.out.println("Enter file name to send");

fileName = scan.nextLine(); file = new File(fileName);

fileInputStream = new FileInputStream(file); int fileLength = (int) file.length();

bytes = new byte[fileLength]; fileInputStream.read(bytes, 0, bytes.length); dataOutputStream.writeUTF(file.getName()); outputStream.write(bytes, 0, bytes.length); outputStream.close();

scan.close();

System.out.println("Sent Successfully"); serverSocket.close();

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

}

}

}

Output:

# Program 4. Implement a chat server using socket programming.

**Chat Client -**

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

class ChatClient{

public static void main(String[] args){ Socket socket = null;

Scanner scan = new Scanner(System.in); String yourMsg, serverMsg; InputStream inputStream = null; OutputStream outputStream = null;

DataInputStream dataInputStream = null; DataOutputStream dataOutputStream = null; try{

socket = new Socket("localhost", 7313); System.out.println("Connected..."); inputStream = socket.getInputStream(); outputStream = socket.getOutputStream();

dataInputStream = new DataInputStream(inputStream); dataOutputStream = new DataOutputStream(outputStream); while(true){

serverMsg = dataInputStream.readUTF(); if(serverMsg.equals("exit")){

break;

}

System.out.println("Server: " + serverMsg); System.out.print("You: ");

yourMsg = scan.nextLine(); dataOutputStream.writeUTF(yourMsg); if(yourMsg.equals("exit")){

break;

}

}

socket.close();

}catch(IOException e){ System.out.println(e);

}

}

}

**Chat Server -**

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

class ChatServer{

public static void main(String[] args){ ServerSocket serverSocket = null; Socket client = null;

Scanner scan = new Scanner(System.in); String yourMsg, clientMsg; OutputStream outputStream = null; InputStream inputStream = null;

DataOutputStream dataOutputStream = null; DataInputStream dataInputStream = null; try {

serverSocket = new ServerSocket(7313); System.out.println("Server started..."); client = serverSocket.accept(); System.out.println("Client connected..."); outputStream = client.getOutputStream(); inputStream = client.getInputStream();

dataOutputStream = new DataOutputStream(outputStream); dataInputStream = new DataInputStream(inputStream); dataOutputStream.writeUTF("Hi from server"); while(true){

clientMsg = dataInputStream.readUTF(); if(clientMsg.equals("exit")){

break;

}

System.out.println("Client: " + clientMsg); System.out.print("You: ");

yourMsg = scan.nextLine(); dataOutputStream.writeUTF(yourMsg); if(yourMsg.equals("exit")){

break;

}

}

serverSocket.close();

}catch(IOException e){ System.out.println(e);

}

}

}

Output:

# Program 5. Implement an ECHO server using socket programming.

**ECHO Client -**

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

class Client{

public static void main(String[] args){ Socket socket = null;

Scanner scan = new Scanner(System.in); String yourMsg, echo;

InputStream inputStream = null; OutputStream outputStream = null; DataInputStream dataInputStream = null; DataOutputStream dataOutputStream = null; try{

socket = new Socket("localhost", 7313); System.out.println("Connected...");

inputStream = socket.getInputStream(); outputStream = socket.getOutputStream();

dataInputStream = new DataInputStream(inputStream); dataOutputStream = new DataOutputStream(outputStream);

System.out.println("Enter msg to echo..."); yourMsg = scan.nextLine();

dataOutputStream.writeUTF(yourMsg); echo = dataInputStream.readUTF();

System.out.println(echo); System.out.println("Echoed successfully"); socket.close();

scan.close();

}catch(IOException e){ System.out.println(e);

}

}

}

**ECHO Server -**

import java.net.\*; import java.io.\*;

class EchoServer{

public static void main(String[] args){ ServerSocket serverSocket = null; Socket client = null;

String clientMsg;

InputStream inputStream = null; OutputStream outputStream = null;

DataInputStream dataInputStream = null; DataOutputStream dataOutputStream = null; try

{

serverSocket = new ServerSocket(7313); System.out.println("Server started..."); client = serverSocket.accept(); System.out.println("Client connected...");

inputStream = client.getInputStream(); outputStream = client.getOutputStream();

dataInputStream = new DataInputStream(inputStream); dataOutputStream = new DataOutputStream(outputStream);

clientMsg = dataInputStream.readUTF(); System.out.println("Echoing..."); dataOutputStream.writeUTF(clientMsg);

serverSocket.close();

}catch(IOException e){ System.out.println(e);

}

}

}

Output:

# Program 6. Implement Address Resolution Protocol using socket programming.

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

public class ARPDemo {

public static void main(String[] args) { String ip;

Scanner scan = new Scanner(System.in); ProcessBuilder processBuilder = new ProcessBuilder(); Process process;

System.out.println("Enter the ip address"); ip = scan.nextLine();

InputStream is; try{

InetAddress inet = InetAddress.getByName(ip); if(inet.isReachable(5000)){

process = processBuilder.command("arp", "-a").start(); is = process.getInputStream();

BufferedReader buff = new BufferedReader(new InputStreamReader(is)); String res;

while((res = buff.readLine()) != null){ if(res.contains(ip)){

res = res.trim();

res = res.replaceAll(" +", " "); String[] array = res.split(" ");

System.out.println(array[0] + " ==> " + array[1]);

}

}

}else{

System.out.println("Host is not present");

}

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

}

}

}

Output:

# Program 7. Implement Ping server and Ping client using socket programming.

**Ping Client –**

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

public class PingClient {

public static void main(String[] args){ DatagramSocket socket; DatagramPacket packet; InetAddress ipaddr;

byte[] bytes; int n = 0; String str; long sTime; long rTime; long cTime; try{

socket = new DatagramSocket(7312);

ipaddr = InetAddress.getByName("localhost"); while(n < 5){

str = "dummy packet"; bytes = str.getBytes();

packet=new DatagramPacket(bytes,bytes.length,ipaddr,7313); socket.send(packet);

sTime = new Date().getTime(); try{

byte[] rBytes = new byte[1024];

DatagramPacket rPacket = new DatagramPacket(rBytes, 0, rBytes.length); socket.setSoTimeout(5000); socket.receive(rPacket); rTime = new Date().getTime();

cTime = rTime - sTime;

System.out.println("Reply from " + rPacket.getAddress().toString() + ": time < " + cTime + "ms");

}catch(IOException ex){ System.out.println("Request Timeout: " + n);

}

Thread.sleep(1000); n++;

}

} catch (SocketException ex) { System.out.println(ex.getMessage());

} catch (UnknownHostException ex) { System.out.println(ex.getMessage());

} catch (IOException ex) { System.out.println(ex.getMessage());

} catch (InterruptedException ex) { System.out.println(ex.getMessage());

}

}

}

**Ping Server –**

import java.net.\*; import java.io.\*;

public class PingServer {

public static void main(String[] args){ DatagramSocket sock; DatagramPacket packet;

byte[] rBytes = new byte[1024]; String msg;

String str; int n = 0; try {

sock = new DatagramSocket(7313); System.out.println("Server started");

packet = new DatagramPacket(rBytes, 0, rBytes.length); while(n < 5){

sock.receive(packet);

msg = new String(rBytes); System.out.println(packet.getAddress().toString() + ": is Pinging"); byte[] sBytes;

str = "dummy packet"; sBytes = str.getBytes();

DatagramPacket sPacket = new DatagramPacket(sBytes,0, sBytes.length, InetAddress.getByName("localhost"), 7312);

sock.send(sPacket); n++;

}

} catch (SocketException ex) { System.out.println(ex.getMessage());

} catch (IOException ex) { System.out.println(ex.getMessage());

}

}

}

Output:

# Program 8. Implement Remote Command Execution using network programming.

**RCE Client –**

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

public class RCEClient {

public static void main(String[] args){ Socket client;

InputStream is; OutputStream os; DataOutputStream dos; DataInputStream dis;

Scanner scan = new Scanner(System.in); String cmd;

try{

client = new Socket("localhost", 7313); System.out.println("Connected to server: " + client.getInetAddress()); System.out.println("Enter the command to execute remotely");

cmd = scan.nextLine();

is = client.getInputStream(); os = client.getOutputStream();

dis = new DataInputStream(is); dos = new DataOutputStream(os); dos.writeUTF(cmd); System.out.println(dis.readUTF()); os.flush();

client.close();

}catch(IOException e){ System.out.println(e.getMessage());

}

}

}

**RCE Server –**

import java.net.\*; import java.io.\*;

class RCEServer {

public static void main(String[] args) { ServerSocket server;

Socket client;

OutputStream os;

InputStream is;

InputStream commandInputStream; DataInputStream dis; DataOutputStream dos;

String[] command; Process process;

ProcessBuilder pB = new ProcessBuilder(); try{

server = new ServerSocket(7313); System.out.println("Server started..."); client = server.accept();

System.out.println("Client connected: " + client.getInetAddress()); os = client.getOutputStream();

is = client.getInputStream(); dis = new DataInputStream(is);

dos = new DataOutputStream(os); String cmd = dis.readUTF();

cmd = cmd.trim(); command = cmd.split(" ");

process = pB.command(command).start(); commandInputStream = process.getInputStream(); BufferedReader buff = new BufferedReader(new

InputStreamReader(commandInputStream)); String read;

String result = "";

while ((read = buff.readLine()) != null){ result = read + "\n" + result;

}

dos.writeUTF(result); dos.flush();

System.out.println("Command: " + cmd + " successfully."); System.out.println("Server exited");

server.close();

}catch(IOException e){

System.out.println(e.getMessage());

}

}

}

Output:

# Program 9. Implement a program to retrieve the data for the specified URL.

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

public class RetrieveData {

public static void main(String[] args) { Scanner scan = new Scanner(System.in); HttpURLConnection connection;

URL url; String input; try {

System.out.println("Enter the URL"); input = scan.nextLine();

url = new URL(input);

connection = (HttpURLConnection) url.openConnection(); System.out.println("Request Method: " + connection.getRequestMethod()); System.out.println("Response Code: " + connection.getResponseCode()); System.out.println("Response Message: " + connection.getResponseMessage()); Map<String, List<String>> headerFields = connection.getHeaderFields(); Set<String> headerKeys = headerFields.keySet();

for(String key: headerKeys){

System.out.println("Key: " + key + " : " + "Value: " + headerFields.get(key));

}

connection.disconnect(); scan.close();

} catch (Exception ex) { System.out.println(ex);

}

}

}

Output:

# Program 10. Write a Java program to check whether the given DNS is found in the internet or not.

import java.net.\*; import java.util.\*;

class DNSTest{

public static void main(String[] args){ String host = new String();

Scanner input = new Scanner(System.in); InetAddress inetAddress;

try{

System.out.println("Enter host name"); host = input.nextLine();

inetAddress = InetAddress.getByName(host); System.out.println("Host Name: " + inetAddress.getHostName()); System.out.println("Host Address: " +

inetAddress.getHostAddress());

System.out.println();

}catch(UnknownHostException e){ System.out.println("Host not found: " + host);

}

}

}

Output:

# Program 11. Write a network program using HTTP to print the document for the given URL.

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

public class HttpDocumentPrinter { public static void main(String[] args) {

Scanner scan = new Scanner(System.in); HttpURLConnection connection;

URL url;

InputStream inputStream; String input;

try {

System.out.println("Enter the URL"); input = scan.nextLine();

url = new URL(input);

connection = (HttpURLConnection) url.openConnection(); inputStream = connection.getInputStream();

int read;

while((read = inputStream.read()) > -1){ char ch = (char) read; System.out.print(ch);

}

scan.close();

} catch (Exception ex) { System.out.println(ex);

}

}

}

Output:

PART - B

# 1. Simulate a three nodes point â€“ to â€“ point network with duplex links between them. Set the queue size and vary the bandwidth and find the number of packets dropped.

# create a new simulator

set ns [new Simulator]

# open trace and NAM trace file in write mode

set tf [open out.tr w]

$ns trace-all $tf

set nf [open out.nam w]

$ns namtrace-all $nf

##### Decide a topology #########

#

# [udp][cbr]

# [0]------

# | [null]

# [2]------[3]

# |

# [1]------

# [udp][cbr]

#

#################################

# create 4 nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

# create duplex links between nodes

$ns duplex-link $n0 $n2 10Mb 300ms DropTail

$ns duplex-link $n1 $n2 10Mb 300ms DropTail

$ns duplex-link $n2 $n3 1Mb 300ms DropTail

# set up queue size

$ns queue-limit $n0 $n2 10

$ns queue-limit $n1 $n2 10

$ns queue-limit $n2 $n3 10

# setup udp connection for transport layer

set udp0 [new Agent/UDP]

set udp1 [new Agent/UDP]

$ns attach-agent $n0 $udp0

$ns attach-agent $n1 $udp1

# setup cbr(constant bit rate) over udp connection for application layer

set cbr0 [new Application/Traffic/CBR]

set cbr1 [new Application/Traffic/CBR]

$cbr0 attach-agent $udp0

$cbr1 attach-agent $udp1

# set n3 as destination node

set null3 [new Agent/Null]

$ns attach-agent $n3 $null3

# connect source to destination

$ns connect $udp0 $null3

$ns connect $udp1 $null3

# set bandwidth (vary values for different output)

$cbr0 set packetSize\_ 500Mb

$cbr1 set packetSize\_ 500Mb

$cbr0 set interval\_ 0.005

$cbr1 set interval\_ 0.005

# define a finish procedure

proc finish {} {

global ns nf tf

$ns flush-trace

exec nam out.nam &

close $tf

close $nf

set count 0

set tf [open out.tr r]

while {[gets $tf line] != -1} {

# d is event in the trace file which denotes dropped packets

if { [string match "d\*" $line] } {

set count [expr $count + 1]

}

}

puts "Number of packets dropped: $count"

exit 0

}

# schedule events

$ns at 0.01 "$cbr0 start"

$ns at 0.01 "$cbr1 start"

$ns at 5.0 "finish"

$ns run

OUTPUT :

# 2. Simulate the different types of Internet traffic such as FTP and TELNET over a network and analyze the throughput.

# create a new simulator

set ns [new Simulator]

# open trace and NAM trace file in write mode

set tf [open out.tr w]

$ns trace-all $tf

set nf [open out.nam w]

$ns namtrace-all $nf

######## Decide a topology #######

#

# [ftp]

# [tcp]

# [0]------

# | [sink0]

# [2]------[3]

# | [sink1]

# [1]------

# [tcp]

# [telnet]

#

##################################

# create 4 nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

# create duplex links between nodes

$ns duplex-link $n0 $n2 2Mb 1ms DropTail

$ns duplex-link $n1 $n2 2Mb 1ms DropTail

$ns duplex-link $n2 $n3 2Mb 1ms DropTail

# set n0 and n1 as tcp source

set tcp0 [new Agent/TCP]

set tcp1 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

$ns attach-agent $n1 $tcp1

# set n3 as tcp destination for n0 and n1

set TCPS0 [new Agent/TCPSink]

set TCPS1 [new Agent/TCPSink]

$ns attach-agent $n3 $TCPS0

$ns attach-agent $n3 $TCPS1

# set ftp over tcp0

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

#set telnet over tcp1

set tel1 [new Application/Telnet]

$tel1 attach-agent $tcp1

$tel1 set packetSize\_ 500Mb

$tel1 set interval\_ 0.001

# connect source to destination

$ns connect $tcp0 $TCPS0

$ns connect $tcp1 $TCPS1

proc finish { } {

global ns nf tf

$ns flush-trace

exec nam out.nam &

close $tf

close $nf

# because time difference between start and finish is 2

set time 2

set fCount 0

set tCount 0

set tf [open out.tr r]

while {[gets $tf line] != -1} {

if { [string match "\*tcp\*0.0\*3.0\*" $line] } {

set fCount [expr $fCount + 1]

}

if { [string match "\*tcp\*1.0\*3.1\*" $line] } {

set tCount [expr $tCount + 1]

}

}

puts "Throughput of FTP: [expr $fCount/$time]"

puts "Throughput of TELNET: [expr $tCount/$time]"

exit 0

}

# schedule events

$ns at 0.01 "$ftp0 start"

$ns at 0.01 "$tel1 start"

$ns at 2.01 "finish"

$ns run

OUTPUT :

# 3. Simulate an Ethernet LAN using n nodes (6-10), change error rate and data rate and compare the throughput.

# Declare a new Simulator

set ns [new Simulator]

# Open nam and trace file in write mode

set tf [open out.tr w]

set nf [open out.nam w]

$ns trace-all $tf

$ns namtrace-all $nf

# Take value of error rate and data rate from std input

puts "Enter error rate (<1) : "

gets stdin erate

puts "Enter data rate (in Mbps) : "

gets stdin drate

############## Select a topology #####################

#

# [udp1] duplex-link

#. [n0] [n1] [n2] [n3]-------------

# | | | | |

# | | | | |

# -------------------------------lan7 |

# |

# -------------------------------lan8 |

# | | | |

# | | | |

# [n4] [n5] [n6]----------------

# [null5]

#

#####################################################

# 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]

set n6 [$ns node]

# set label and color (OPTIONAL)

$n1 label "udp/source"

$n5 label "udp/null"

$n0 color "blue"

$n1 color "blue"

$n2 color "blue"

$n3 color "blue"

$n4 color "red"

$n5 color "red"

$n6 color "red"

# Create two lans

$ns make-lan "$n0 $n1 $n2 $n3" 10Mb 10ms LL Queue/DropTail Mac/802\_3

$ns make-lan "$n4 $n5 $n6" 10Mb 10ms LL Queue/DropTail Mac/802\_3

# Setup Links

$ns duplex-link $n3 $n6 10Mb 10ms DropTail

# Declare the transport layer protocols

set udp1 [new Agent/UDP]

set null5 [new Agent/Null]

$ns attach-agent $n1 $udp1

$ns attach-agent $n5 $null5

# Declare the application layer protocol

set cbr1 [new Application/Traffic/CBR]

$cbr1 attach-agent $udp1

# Connect the source and destination

$ns connect $udp1 $null5

# Create error model

set err [new ErrorModel]

$ns lossmodel $err $n3 $n6

$err set rate\_ $erate

# Define the data rate

$cbr1 set packetSize\_ $drate.Mb

$cbr1 set interval\_ 0.001

# Define procedure

proc finish { } {

global ns nf tf

$ns flush-trace

exec nam out.nam &

close $nf

close $tf

set count 0

set tr [open out.tr r]

while {[gets $tr line] != -1} {

# 8 denotes LAN at destination side and 5 denotes destination node

if {[string match "\* 8 5 \*" $line]} {

set count [expr $count+1]

}

}

set thr [expr $count/5]

puts "Throughput : $thr"

exit 0

}

$ns at 0.1 "$cbr1 start"

$ns at 5.1 "finish"

$ns run

OUTPUT :

# 4. Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and determine the collision across different nodes.

# Declare a new Simulator

set ns [new Simulator]

# Open the trace and nam file in write mode

set tf [open out.tr w]

set nf [open out.nam w]

$ns trace-all $tf

$ns namtrace-all $nf

# Decide the topology: [tcp(0->2)], [udp(2->1)], [tcp(1->3)]

#

# [tcp0] [tcp1][null1]

# [n0] [n1]

# | |

# | |

# ------------------------------ lan4

# | |

# | |

# [n3] [n2]

# [sink3] [udp2][sink2]

# Create 4 nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

# Create lan and setup the link

$ns make-lan -trace on "$n0 $n1 $n2 $n3" 100Mb 10ms LL Queue/DropTail Mac/802\_3

# Declare the required transport layer Protocols

set tcp0 [new Agent/TCP]

set tcp1 [new Agent/TCP]

set udp2 [new Agent/UDP]

set null1 [new Agent/Null]

set sink2 [new Agent/TCPSink]

set sink3 [new Agent/TCPSink]

# Attach these Protocols to their respective nodes

$ns attach-agent $n0 $tcp0

$ns attach-agent $n1 $tcp1

$ns attach-agent $n2 $udp2

$ns attach-agent $n1 $null1

$ns attach-agent $n2 $sink2

$ns attach-agent $n3 $sink3

# Declare Application layer protocols and attach them with their transport layer protocols

set ftp0 [new Application/FTP]

set ftp1 [new Application/FTP]

set cbr2 [new Application/Traffic/CBR]

$ftp0 attach-agent $tcp0

$ftp1 attach-agent $tcp1

$cbr2 attach-agent $udp2

# connect source to destination

$ns connect $tcp0 $sink2

$ns connect $udp2 $null1

$ns connect $tcp1 $sink3

# set the interval

$ftp0 set interval\_ 0.001

$ftp1 set interval\_ 0.001

$cbr2 set interval\_ 0.01

# define finish procedure

proc finish {} {

global ns nf tf

$ns flush-trace

exec nam out.nam &

close $tf

close $nf

set count 0

set tr [open out.tr r]

while {[gets $tr line] !=-1 } {

if { [string match "c\*" $line] } {

set count [expr $count + 1]

}

}

puts "No of packets collided: $count"

exit 0

}

# schedule the events

$ns at 0.1 "$cbr2 start"

$ns at 0.1 "$ftp0 start"

$ns at 0.1 "$ftp1 start"

$ns at 5.0 "finish"

$ns run

OUTPUT :