**CNC - MANUAL**

**PART-A**

1. **Simulate a three nodes point-to-point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.**

set ns [new Simulator]

set nf [open out.nam w]

$ns namtrace-all $nf

set tf [open out.tr w]

$ns trace-all $tf

proc finish {} {

global ns nf tf

$ns flush-trace

close $nf

close $tf

exec nam out.nam &

exit 0

}

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

$ns duplex-link $n0 $n1 1Mb 10ms DropTail

$ns duplex-link $n1 $n2 1Mb 10ms DropTail

$ns queue-limit $n0 $n1 50

$ns queue-limit $n1 $n2 50

set udp0 [new Agent/UDP]

$ns attach-agent $n0 $udp0

set null0 [new Agent/Null]

$ns attach-agent $n2 $null0

$ns connect $udp0 $null0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.005

$cbr0 attach-agent $udp0

$ns at 0.5 "$cbr0 start"

$ns at 4.5 "$cbr0 stop"

$ns at 5.0 "finish"

$ns run

AWK

BEGIN{

count=0;

}

{

event=$1;

if(event=="d")

{

count++;

}

}

END{

printf("\nNumber of packets dropped is :%d\n",count);

}

1. **Simulate a 4 node point to point network and connect the link as follows n1-n2, n2-n3, and n4-n2. Apply TCP agent between a relevant application over tcp and udp.**
   * **Determine the number of packets by tcp and udp**
   * **Number of packets dropped during the transmission**
   * **Analyze the throughput by varying the network parameters**

set ns [new Simulator]

set nf [open out.nam w]

$ns namtrace-all $nf

set tf [open out.tr w]

$ns trace-all $tf

proc finish {} {

global ns nf tf

$ns flush-trace

close $nf

close $tf

exec nam out.nam &

exit 0

}

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

$ns duplex-link $n0 $n2 1Mb 10ms DropTail

$ns duplex-link $n1 $n2 1Mb 10ms DropTail

$ns duplex-link $n2 $n3 1Mb 10ms DropTail

$ns queue-limit $n0 $n2 50

$ns queue-limit $n1 $n2 50

$ns queue-limit $n2 $n3 50

set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n3 $sink0

$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

Agent/TCP set packetSize\_ 1000

set udp0 [new Agent/UDP]

$ns attach-agent $n1 $udp0

set null0 [new Agent/Null]

$ns attach-agent $n3 $null0

$ns connect $udp0 $null0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.005

$cbr0 attach-agent $udp0

$ns at 0.75 "$ftp0 start"

$ns at 4.75 "$ftp0 stop"

$ns at 0.5 "$cbr0 start"

$ns at 4.5 "$cbr0 stop"

$ns at 5.0 "finish"

$ns run

AWK

BEGIN{

cupd=0;

ctcp=0;

count=0;

}

{

if($5 == "cbr"){cudp++;}

if($5 == "tcp"){ctcp++;}

if($1 == "d"){count++;}

}

END{

printf("Number of TCP packets= %d\n",ctcp);

printf("Number of UDP packets= %d\n",cudp);

printf("Number of packets dropped= %d\n",count);

}

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

set ns [new Simulator]

set nf [open out.nam w]

$ns namtrace-all $nf

set tf [open out.tr w]

$ns trace-all $tf

proc finish {} {

global ns nf tf

$ns flush-trace

close $nf

close $tf

exec nam out.nam &

exit 0

}

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

$ns duplex-link $n0 $n2 1Mb 10ms DropTail

$ns duplex-link $n1 $n2 1Mb 10ms DropTail

$ns duplex-link $n2 $n3 1Mb 10ms DropTail

$ns queue-limit $n0 $n2 50

$ns queue-limit $n1 $n2 50

$ns queue-limit $n2 $n3 50

set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n3 $sink0

$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

Agent/TCP set packetSize\_ 1000

set tcp1 [new Agent/TCP]

$ns attach-agent $n1 $tcp1

set sink1 [new Agent/TCPSink]

$ns attach-agent $n3 $sink1

$ns connect $tcp1 $sink1

set telnet0 [new Application/Telnet]

$telnet0 set interval\_ 0.005

$telnet0 attach-agent $tcp1

$ns at 0.75 "$ftp0 start"

$ns at 4.75 "$ftp0 stop"

$ns at 0.5 "$telnet0 start"

$ns at 4.5 "$telnet0 stop"

$ns at 5.0 "finish"

$ns run

AWK

BEGIN {

sSize = 0;

startTime = 5.0;

stopTime = 0.1;

Tput = 0;

}

{

event = $1;

time = $2;

size = $6;

if(event == "+")

{

if(time < startTime)

{

startTime = time;

}

}

if(event == "r")

{

if(time > stopTime)

{

stopTime = time;

}

sSize += size;

}

Tput = (sSize / (stopTime-startTime))\*(8/1000);

printf("%f\t%.2f\n", time, Tput);

}

END {

}

1. **Simulate an Ethernet LAN using N nodes and set multiple traffic nodes and determine collision across different nodes.**

set ns [new Simulator]

set nf [open out.nam w]

$ns namtrace-all $nf

set tf [open out.tr w]

$ns trace-all $tf

proc finish {} {

global ns nf tf

$ns flush-trace

close $nf

close $tf

exec nam out.nam &

exit 0

}

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 n7 [$ns node]

set n8 [$ns node]

set n9 [$ns node]

$ns make-lan "$n0 $n1 $n2 $n3 $n4 $n5 $n6 $n7 $n8 $n9" 100Mb 10ms LL

Queue/DropTail Mac/802\_3

set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCPSink]

$ns attach-agent $n3 $sink0

$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

Agent/TCP set packetSize\_ 1000

set tcp1 [new Agent/TCP]

$ns attach-agent $n1 $tcp1

set sink1 [new Agent/TCPSink]

$ns attach-agent $n3 $sink1

$ns connect $tcp1 $sink1

set telnet0 [new Application/Telnet]

$telnet0 set interval\_ 0.005

$telnet0 attach-agent $tcp1

$ns at 0.75 "$ftp0 start"

$ns at 4.75 "$ftp0 stop"

$ns at 0.5 "$telnet0 start"

$ns at 4.5 "$telnet0 stop"

$ns at 5.0 "finish"

$ns run  
  
AWK

BEGIN{

count=0;

}

{

event=$1;

if(event=="d")

{

count++;

}

}

END{

printf("\nNumber of packets dropped is :%d\n",count);

}

**PART-B**

1. **Write a program for frame sorting technique used in buffers.**

import random as rd

msg=input("Enter the message :")

packets = [[i//3,msg[i:(i+3) if i+3 < len(msg) else None]] for i in range(0,len(msg),3)]

print("packets :",packets)

rd.shuffle(packets)

print("shuffled packets: ",packets)

packets.sort(key=lambda a: a[0])

print("sorted packets: ",packets)

1. **Write a program for distance vector algorithm to find suitable path for transmission.**

#include<stdio.h>

struct node

{

int dist[20];

int from[20];

}rt[10];

int main()

{

int dmat [20] [20];

int n, i, j, k, count=1;

printf ("\nEnter the number of nodes :\n");

scanf ("%d", &n);

printf ("\nEnter the cost matrix :\n");

for (i=1; i<=n; i++)

for (j=1; j<=n; j++)

{

scanf ("%d", &dmat[i][j]);

dmat [i][i] = 0;

rt[i].dist[j] = dmat[i][j];

rt[i].from[j] = j;

}

do

{

for (i=1; i<=n; i++)

for (j=1; j<=n; j++)

for (k=1; k<=n; k++)

if (rt[i].dist[j] > dmat[i][k] + rt[k].dist[j])

{

rt[i].dist[j] = rt[i].dist[k] + rt[k].dist[j];

rt[i].from[j] = k;

}

count++;

}while (count < n);

for (i=1; i<=n; i++)

{

printf ("\nDistance Table for router %c is \n", i+64);

for (j=1; j<=n; j++)

printf ("\tNode %d Via %d, Distance : %d\n", j, rt[i].from[j], rt[i].dist[j]);

}

return 0;

}

1. **Using UDP SOCKETS, write a client-server program to make the client sending two numbers and an operator, and server responding with the result. Display the result and appropriate messages for invalid inputs at the client side.**

**UDP-Server**

import socket

localIP = "127.0.0.1"

localPort = 20002

bufferSize = 1024

# Create a datagram socket

UDPServerSocket = socket.socket(family=socket.

# Bind to address and ip

UDPServerSocket.bind((localIP, localPort))

print("UDP server up and listening...")

# Listen for incoming datagrams

while(True):

bytesAddressPair = UDPServerSocket.recvfrom(

expression = bytesAddressPair[0].decode()

client\_address = bytesAddressPair[1]

if expression:

result=eval(expression)

else:

result = 'Invalid expression.'

print("Equation from client", client\_address, ":", expression)

print("Result to client:", result)

# Sending a reply to client

UDPServerSocket.sendto(str(

UDPServerSocket.close()

**UDP-Client**  
import socket

expr = input("Enter an expression:")

bytesToSend = str.encode(expr)

serverAddressPort = ("127.0.0.1", 20001)

bufferSize = 1024

# Create a UDP socket at client side

UDPClientSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

# Send to server using created UDP socket

UDPClientSocket.sendto(bytesToSend, serverAddressPort)

msgFromServer = UDPClientSocket.recvfrom(bufferSize)

print(expr, "=", msgFromServer[0].decode())