CNC LAB 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.**

prgrm.tcl

```
#create simulator object
set ns [new Simulator]

#open trace file
set f [open out.tr w]
$ns trace-all $f

#open nam file
set nf [open out.nam w]
$ns namtrace-all $nf

#create 3 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

#create duplex-link between the nodes
$ns duplex-link $n0 $n1 0.1Mb 10ms DropTail
$ns duplex-link $n1 $n2 0.1Mb 10ms DropTail

#set queue-limit between the nodes
$ns queue-limit $n0 $n1 50
$ns queue-limit $n1 $n2 50

#define a 'finish' procedure
proc finish {} {
```

```
        global ns f nf
        $ns flush-trace
        close $f
        exec nam out.nam &
        close $nf
        exit 0
}

#add UDP agent
# UDP(User Datagram Protocol)
 set udp [new Agent/UDP]
 $ns attach-agent $n0 $udp

 #add null agent
 set null [new Agent/Null]
 $ns attach-agent $n2 $null

# Establish UDP connection to null
$ns connect $udp $null

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

$cbr set packetSize_ 500
$cbr set interval_ 0.008

#labeling the nodes
$n0 label "src"
$n1 label "dest"

#start and stop the data transmission
$ns at 3.0 "$cbr start"
$ns at 4.5 "$cbr stop"

#call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#run the simulation
$ns run
```

p.awk

```
BEGIN{
        count=0;
}
{
        event=$1;
        if(event=="d")
        {
        count++;
        }
}
END{
        printf("\nNumber of packets dropped is :%d\n",count);
}
```

2. **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
   -> Analyse the throughput by varying the network parameters

   Prgrm.tcl

```
#create simulator
set ns [new Simulator]

#trace file
set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf

#finish procedure
proc finish {} {
global ns nf tf
$ns flush-trace
close $tf
close $nf
```

```
exec nam out.nam &
#exec awk -f prog2.awk out.tr &
#exec awk -f thrp.awk out.tr &
exit 0
}

#create the nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

#label
$n1 label "src"
$n4 label "udpsrc"
$n3 label "dest"

#create the links & assign the bandwith and delay
$ns duplex-link $n1 $n2 1mb 10ms DropTail
$ns duplex-link $n2 $n3 0.005mb 10ms DropTail
$ns duplex-link $n2 $n4 1.5mb 10ms DropTail
#set the Queue-limit
$ns set queue-limit $n1 $n2 50
$ns set queue-limit $n2 $n3 5
$ns set queue-limit $n2 $n4 5

#create TCP
set tcp [new Agent/TCP]
$ns attach-agent $n1 $tcp
#create TCPSInk
set tcpSink [new Agent/TCPSink]
$ns attach-agent $n3 $tcpSink
#conect both
$ns connect $tcp $tcpSink

#create FTP (traffic )
set ftp [new Application/FTP]
$tcp set packetSize_ 500
#attch ftp and tcp
$ftp attach-agent $tcp

#create the Transport agent
set udp [new Agent/UDP]
$ns attach-agent $n4 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null

#create CBR application
set cbr [new Application/Traffic/CBR]
```

```
$cbr set packetSize_ 700
$cbr set interval_ 0.05
$cbr attach-agent $udp
#$udp set fid_ 1
#$udp set class_ 1

#colour
$ns color 1 "blue"
$ns color 2 "red"

$tcp set class_ 1
$udp set class_ 2

#start & stop the Data transmition
$ns at 0.5 "$cbr start"
$ns at 4.5 "$cbr stop"

$ns at 1.5 "$ftp start"
$ns at 5.5 "$ftp stop"

#create the finish procedure
$ns at 6.0 "finish"

#Run the Simulation
$ns run
```

p.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);
}
```

thrp.awk

```
BEGIN{
  stime=0
  ftime=0
  flag=0
  fsize=0
```

```
  tp=0
  latency=0
}
{
  if($1 == "r" && $4 == 2)
  {
    fsize+=$6
    if(flag == 0)
    {
      stime=$2
      flag=1
    }
    ftime=$2
  }
}
END{
  latency=ftime-stime
  tp=(fsize*8)/latency
  printf("\nlatency of the network is : %d\n",latency);
  printf("\nThoughput of the network is : %d\n",tp);
}
```

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

prgrm.tcl

```
#create simulator
set ns [new Simulator]

#trace file
set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf

#finish procedure
proc finish {} {
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam out.nam &
#exec awk -f thrp.awk out.tr &
exit 0
}

#create the 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]

#label
$n1 label "SRC"
$n5 label "DST"

#create a LAN
$ns make-lan "$n0 $n1 $n2 $n3 $n4 $n6" 10Mb 10ms LL Queue/DropTail Mac/802_3

#connect n4 and n5
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns queue-limit $n4 $n5 10
$ns duplex-link-op $n4 $n5 orient right-down

#create TCP
set tcp [new Agent/TCP]
$ns attach-agent $n1 $tcp
#create TCPSInk
set tcpSink [new Agent/TCPSink]
$ns attach-agent $n5 $tcpSink

#conect both
$ns connect $tcp $tcpSink

#create FTP (traffic )
set ftp [new Application/FTP]

#attch ftp and tcp
$ftp attach-agent $tcp

#start & stop the Data transmition
$ns at 0.5 "$ftp start"
$ns at 5.0 "$ftp stop"

#create the finish procedure
$ns at 5.2 "finish"

#Run the Simulation
$ns run
```

lan.awk

```
BEGIN{
  sSize = 0;
  startTime = 5.0;
  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{
}
```

4. **Stimulate an ethernet LAN with four node. Assign multiple such as Telnet. Determine the army of network by varying data net**

prgrm.tcl

```
#create simulator
set ns [new Simulator]

#trace file
set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf

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

#create the 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]

#label
$n0 label "SRC1"
$n3 label "DST2"
$n4 label "SRC2"
$n6 label "DST2"

#create a LAN
$ns make-lan "$n0 $n1 $n2 $n3" 10Mb 10ms LL Queue/DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n2 $n6" 10Mb 10ms LL Queue/DropTail Mac/802_3

#create TCP1
set tcp1 [new Agent/TCP]
$ns attach-agent $n0 $tcp1
#create TCPSInk1
set tcpSink1 [new Agent/TCPSink]
$ns attach-agent $n6 $tcpSink1

#conect both
$ns connect $tcp1 $tcpSink1

#create FTP (traffic )
set ftp [new Application/FTP]
#attch ftp and tcp
$ftp attach-agent $tcp1
$ftp set interval_ 0.5

#create TCP2
set tcp2 [new Agent/TCP]
$ns attach-agent $n4 $tcp2

#create TCPSInk2
set tcpSink2 [new Agent/TCPSink]
$ns attach-agent $n3 $tcpSink2

#conect both
$ns connect $tcp2 $tcpSink2

#create Telnet Application
set telnet [new Application/Telnet]
$telnet attach-agent $tcp2
$telnet set interval_ 0.5

#color
$ns color 1 "blue"
$ns color 2 "red"

$tcp1 set class_ 1
$tcp2 set class_ 2

#start & stop the Data transmition
```

```
$ns at 0.5 "$ftp start"
$ns at 4.5 "$ftp stop"

$ns at 0.3 "$telnet start"
$ns at 3.5 "$telnet stop"

#call the finish procedure
$ns at 5.5 "finish"

#Run the Simulation
$ns run
```

lan.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{
}
```

5. **Simulate an Ethernet LAN using N nodes and set multiple traffic nodes and determine collision across different nodes, also plot congestion window for different source/destination.**

prgrm.tcl

```
#create simulator
set ns [new Simulator]

#trace file
set tf [open out.tr w]
$ns trace-all $tf
set nf [open out.nam w]
$ns namtrace-all $nf

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

#create the 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]

#label
$n1 label "SRC1"
$n3 label "DST1"
$n1 label "Src2"
$n5 label "DST2"

#create a LAN
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 10Mb 10ms LL Queue/DropTail Mac/802_3

#connect n4 and n5
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns queue-limit $n4 $n5 10
$ns duplex-link-op $n4 $n5 orient right-down

#create TCP1
set tcp1 [new Agent/TCP]
$ns attach-agent $n0 $tcp1
#create TCPSInk1
set tcpSink1 [new Agent/TCPSink]
$ns attach-agent $n3 $tcpSink1

#conect both
$ns connect $tcp1 $tcpSink1

#create FTP1 (traffic )
set ftp1 [new Application/FTP]

#attch ftp1 and tcp1
```

```
$ftp1 attach-agent $tcp1

#create TCP2
set tcp2 [new Agent/TCP]
$ns attach-agent $n1 $tcp2
#create TCPSInk2
set tcpSink2 [new Agent/TCPSink]
$ns attach-agent $n5 $tcpSink2
#conect both
$ns connect $tcp2 $tcpSink2

#create FTP2 (traffic )
set ftp2 [new Application/FTP]

#attach ftp2 and tcp2
$ftp2 attach-agent $tcp2

#start & stop the Data transmition
$ns at 0.0 "$ftp1 start"
$ns at 1.0 "$ftp2 start"
$ns at 5.0 "$ftp1 stop"
$ns at 6.0 "$ftp2 stop"
$ns at 6.0 "$ftp1 start"
$ns at 7.0 "$ftp2 start"
$ns at 12.0 "$ftp1 stop"
$ns at 13.0 "$ftp2 stop"

set file1 [open file1.tr w]
$tcp1 attach $file1

set file2 [open file2.tr w]
$tcp2 attach $file2

$tcp1 trace cwnd_
$tcp2 trace cwnd_

#create the finish procedure
$ns at 15.0 "finish"

#Run the Simulation
$ns run
```

cong.awk

```
BEGIN{
}
{
   if($6=="cwnd_")
   {
      print("%f\t\t%f\n",$1,$7);
   }
}
END{
}
```

PART-B

6. Program to implement Bit Stuffing or Byte Stuffing concept in data link layer

```
#include <iostream>
#include <string>
using namespace std;

string stuff(string data) {
    string res;
    int count = 0;

    for (char ch : data) {
        res += ch;
        if (ch == '1') {
            count++;
            if (count == 5) {
                res += '0';
                count = 0;
            }
        } else if (ch == '0') {
            count = 0;
        }
    }

    return res;
}

string unstuff(string data) {
    string res;
    int count = 0;
    for (int i = 0; i < data.size(); ++i) {
        char ch = data[i];
        res += ch;
        if (ch == '1') {
            count++;
            if (count == 5) {
                if (data[i + 1] == '0') {
                    i++;
                }
                count = 0;
            }
        } else if (ch == '0') {
            count = 0;
        }
    }

    return res;
```

```
}


int main() {
    string data;
    cout << "Enter the data to be entered: ";
    cin >> data;

    string stuffed = stuff(data);
    cout << "Stuffed data: " << stuffed << endl;

    string unstuffed = unstuff(stuffed);
    cout << "Unstuffed data: " << unstuffed << endl;

    return 0;
}
```

7. Write a program for error detecting code using CRC-CCITT (16-bits).

```
# program for error detecting code using CRC-CCITT
div =[1,0,0,1]
cdata=[]
n = int(input("Enter the number of bits: "))
data = [int(input(f"Enter bit {i + 1}: ")) for i in range(n)]
m=len(div)

if n<m:
    print("Invalid data ")
else:
    cdata=data.copy()
    for i in range(m-1):
        cdata.append(0)
    print("Initial Codeword :",cdata)
    for i in range(n):
        if cdata[i]==1:
            for j in range(m):
                cdata[i+j]^=div[j]

    for i in range(n,n+m-1):
        data.append(cdata[i])
    print("Codeword to be transmitted is : ",data)
    # Optionally introduce an error
    ch = input("Introduce Error - Y->YES, N->NO: ")
    if ch.upper() == "Y":
        p = int(input("Enter the position to change bit : "))
        if 1 <= p <= len(data):
            data[p - 1] ^= 1  # Flip the bit at the given position
        else:
            print("Invalid Position!")
    print("The message received is:", data)
```

```
for i in range(n):
 if data[i]==1:
     for j in range(m):
         data[i+j]^=div[j]
flag=True

for i in data:
  if i==1:
      flag=False
      break
if(flag):
    print("No Error\nSucessfully Transferred ")
else:
    print("Error!!")
```

8. Write a program for frame sorting technique used in buffers.

```
import random as rd
msg=input("Enter the message :")
frames = [[i//3,msg[i:(i+3) if i+3 < len(msg) else None]] for i in range(0,len(msg),3)]
print("frames :",frames)
rd.shuffle(frames)
print("shuffled frames: ",frames)
frames.sort()
print("sorted frames: ",frames)
```

9. Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

Client_TCP.py

```
import socket

# Define the server address and port
server_address = ('localhost', 8888)

# Create a socket and connect to the server
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(server_address)

# Get the file name from the user
file_name = input("Enter the file name: ")

# Send the file name to the server
client_socket.send(file_name.encode())

# Receive the file contents from the server
file_contents = client_socket.recv(1024).decode()
```

```
if file_contents == b'File not found':
    print("File not found on the server.")
else:
    print(file_contents)
    # Save the received file contents to a new file
   # with open("received_", 'wb') as file:
    #    file.write(file_contents)
    #print("File received and saved as 'received_" + file_name + "'")

# Close the client socket
client_socket.close()
```

Server_TCP.py

```
import socket
import os

# Define the server address and port
server_address = ('localhost', 8888)

# Create a socket and bind it to the server address
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(server_address)

# Listen for incoming connections
server_socket.listen(1)
print("Server is listening for incoming connections...")

while True:
    # Accept a client connection
    client_socket, client_address = server_socket.accept()
    print("Accepted connection from", client_address)

    # Receive the file name from the client
    file_name = client_socket.recv(1024).decode()
    print("Received file name= ",file_name)


    # Check if the file exists
    if os.path.exists(file_name):
        file= open(file_name, 'rb')
            # Read the contents of the file
        file_contents = file.read()
            # Send the file contents to the client
        client_socket.send(file_contents)
    else:
        client_socket.send(b'File not found')

    # Close the client socket
```

```
client_socket.close()
```

10. 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_Client.py

```
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())
```

UDP_Server.py

```
import socket, re

localIP     = "127.0.0.1"
localPort   = 20001
bufferSize  = 1024

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET,
type=socket.SOCK_DGRAM)

# Bind to address and ip
UDPServerSocket.bind((localIP, localPort))

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

```
# Listen for incoming datagrams
while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
    expression       = bytesAddressPair[0].decode()
    client_address   = bytesAddressPair[1]

    num1, num2 = re.split('\+|-|\*|/|%', expression.replace(' ', ''))
    num1 = float(num1)
    num2 = float(num2)

    result = 0
    if '+' in expression:
        result = num1 + num2
    elif '-' in expression:
        result = num1 - num2
    elif '*' in expression:
        result = num1 * num2
    elif '/' in expression:
        if num2 == 0:
            result = 'Division by zero.'
        else:
            result = num1 / num2
    elif '%' in expression:
        result = num1 % num2
    else:
        result = 'Invalid expression.'

    print("Equation from client", client_address, ":", expression)
    print("Result to client:", result)

    # Sending a reply to client
    UDPServerSocket.sendto(str(result).encode(), client_address)
UDPServerSocket.close()
```