CNC

Part - A

Commands:

```
Create a file -> gedit filename.tcl
Run -> ns filename.tcl
For AWK-> awk -f filename.awk out.tr
```

For Graph in 3rd :

```
awk  -f  filename.awk  out.tr > 3a
xgraph  3a
```

Change to 50Mb and 100 ms in TCL code

```
awk -f  filename.awk  out.tr > 3b
xgraph  3b
```

For Graph in 4th :

```
awk -f filename.awk file1.tr > 3a
xgraph  3a
awk -f filename.awk file2.tr > 3b
xgraph  3b
```

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.

   TCL CODE:

```
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 CODE:

```
BEGIN{
   count=0;
}
{
event=$1
src=$9
des=$10
if(event=="d"){
   count++;
}
}
END{
   printf("no of packets dropped from %d to %d is %d",src,des,count);
}
```

2. Simulate a four-node point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets by TCP/UDP and analyze the throughput.

TCL CODE:

```
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 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_ 1000
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

$ns at 0.5 "$ftp0 start"
$ns at 3.5 "$ftp0 stop"

$ns at 3.75 "$cbr0 start"
$ns at 5.0 "$cbr0 stop"
```

```
$ns at 6.0 "finish"

$ns run
```

AWK CODE:

```
BEGIN{
    count=0;
    countr=0;
    ctcp=0;
    cudp=0;
}
{
    event=$1
    packet=$5
    if(event=="d"){
        count++;
    }
    if(event="r"){
        countr++;
    }
    if(packet=="tcp"){
        ctcp++;
    }
    if(packet="cbr"){
        cudp++;
    }
}
END{
    printf("no of packets dropped is %d",count);
     printf("no of packets recieved is %d",countr);
      printf("no of packets dropped by TCP is %d",ctcp);
       printf("no of packets dropped by UDP  is %d",cudp);
}
```

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

TCL CODE:

```
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]
set n10 [$ns node]

$ns make-lan "$n0 $n1 $n2 $n3 $n4 $n5 $n6 $n7 $n8 $n9 $n10" 100Mb 10ms
LLQueue/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

$ns at 0.5 "$ftp0 start"
$ns at 3.5 "$ftp0 stop"

$ns at 4.0 "finish"

$ns run
```

AWK CODE:

```
BEGIN{
    sSize=0;
    startTime=0;
    stopTime=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("%d \t %.2f \t",time,Tput);
}
END{

}
```

4 .Simulate an Ethernet LAN using N nodes and set multiple traffic nodes and determine
collisions across different nodes, also plot congestion windows for different sources/destinations.

TCL CODE:

```
set val(stop) 15.0

set ns [new Simulator]

set tf [open out.tr w]
$ns trace-all $tf

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

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$n0 color "blue"
$n0 label "src1"

$n1 color "blue"
$n1 label "src2"

$n5 color "red"
$n5 label "dest1"

$n3 color "red"
$n3 label "dest2"

$ns make-lan "$n0 $n1 $n2 $n3 $n4 " 10Mb 10ms LL Queue/DropTail Mac/802_3
$ns duplex-link $n4 $n5 1.0Mb 1ms DropTail
$ns queue-limit $n4 $n5 5
```

```
$ns duplex-link-op $n4 $n5 orient right-down

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n5 $sink1
$ns connect $tcp0 $sink1

$tcp0 set packetSize_ 1500

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

$ns at 0.5 "$ftp0 start"
$ns at 5.0 "$ftp0 stop"
$ns at 5.5 "$ftp0 start"
$ns at 12.0 "$ftp0 stop"

set tcp2 [new Agent/TCP]
$ns attach-agent $n1 $tcp2
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp2 $sink3
$tcp0 set packetSize_ 1500

set telnet0 [new Application/Telnet]
$telnet0 set interval_ 0.005
$telnet0 attach-agent $tcp2

$ns at 1.0 "$telnet0 start"
$ns at 6.0 "$telnet0 stop"
$ns at 7.0 "$telnet0 start"
$ns at 13.0 "$telnet0 stop"

set file1 [open file1.tr w]
$tcp0 attach $file1

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

```
$tcp0 trace cwnd_
$tcp2 trace cwnd_

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

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts\"done\"; $ns halt"

$ns run
```

AWK CODE:

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

Part - B

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

```
def sender():
    n=int(input("enter the number of bits:"))
    print("enter",n,"bits")
    data=[int(input()) for _ in range(n)]

    add=[0,1,1,1,1,1,1,0]
    frame=[]

    frame+=add
    count=0

    for i in data:
        if count==5:
            frame.append(0)
            count+=1
        frame.append(i)
        if i==1:
            count+=1
        else:
            count=0

    frame+=add
    print("Sent Frames:",frame)
    reciever(frame)

def reciever(frame):
    data=[]
    count=0

    for i in frame[8:-8]:
        if i==1:
            data.append(i)
            count+=1
        elif i==0:
            if count==5:
                count=0
            else:
                data.append(i)
        else:
            count=0

    print("Data recieved:",data)

sender()
```

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

```
div=[1,0,0,0,1,0]
n=int(input("enter no of bits:"))
data=[int(input()) for _ in range(n)]
m=len(div)

if n<m:
    print("invalid input")
else:
    cdata=data.copy()
    cdata.extend([0]*(m-1))

    for i in range(n):
        if cdata[i]==1:
            for j in range(m):
                cdata[i+j]^= div[j]

    data.extend(cdata[n :n + m - 1])
    print("data transmitted:",data)

    if input("introduce error - Y->Yes N->No")=="Y":
        p=int(input("enter position"))
        data[p-1]^=1

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

    if all(bit==0 for bit in data):
        print("no error")
    else:
        print("error!")
```

3 .Write a program for frame sorting techniques used in buffers.

```
import random
DATA_SZ=3

def bub_sort(arr):
    n=len(arr)
    for i in range(n):
        for j in range(n-i-1):
            if(arr[j]>arr[j+1]):
                arr[j],arr[j+1]=arr[j+1],arr[j]

msg=input("enter message:")
msg_chunks=[msg[i:i+DATA_SZ] for i in range(0,len(msg),DATA_SZ)]
frames=list(enumerate(msg_chunks,start=1))
print("fragmented frames:",frames)

for i in range(len(frames)):
    j=random.randint(0,len(frames)-1)
    frames[i],frames[j]=frames[j],frames[i]


print("Shuffled frames:",frames)
bub_sort(frames)
print("Sorted frames:",frames)
print("Sorted message:"+"".join(x[1] for x in frames))
```

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

```
class Router:
    def __init__(self, name):
        self.name = name
        self.distances = {}
        self.neighbors = {}

    def add_neighbor(self, neighbor, cost):
        self.neighbors[neighbor] = cost
        self.distances[neighbor.name] = cost

    def update_distance(self, neighbor, new_distance):
        if neighbor not in self.distances or new_distance < self.distances[neighbor]:
            self.distances[neighbor] = new_distance

    def __repr__(self):
        return f"{self.name}: {self.distances}"


class Network:
    def __init__(self):
        self.routers = {}

    def add_router(self, router):
        self.routers[router.name] = router
    def update_distance_vectors(self):
        for router in self.routers.values():
            for neighbor, cost in router.neighbors.items():
                for dest, dist in neighbor.distances.items():
                    new_distance = cost + dist
                    router.update_distance(dest, new_distance)

    def display_routing_tables(self):
        for router in self.routers.values():
            print(router)


# Example usage:
if __name__ == "__main__":
    # Create routers
    A = Router("A")
    B = Router("B")
    C = Router("C")
    D = Router("D")

    # Set neighbors and costs
    A.add_neighbor(B, 1)
    A.add_neighbor(C, 4)
    B.add_neighbor(A, 1)
    B.add_neighbor(C, 2)
    B.add_neighbor(D, 5)
    C.add_neighbor(A, 4)
    C.add_neighbor(B, 2)
    C.add_neighbor(D, 1)
    D.add_neighbor(B, 5)
    D.add_neighbor(C, 1)


    network = Network()
    network.add_router(A)
    network.add_router(B)
    network.add_router(C)
    network.add_router(D)


    for _ in range(len(network.routers) - 1):
        network.update_distance_vectors()
```

```
print("Routing tables after convergence:")
network.display_routing_tables()
```