DEPT. OF ELECTRONICS AND COMMUNICATION ENGINEERING

COMPUTER NETWORKS LABORATORY [15ECL68]

MAHARAJA INSTITUTE OF TECHNOLOGY THANDAVAPURA

Nanjangud Taluk, Mysore-571302

INSTRUCTIONS

1. Students must bring observation/Manual book along with pen, pencil and eraser etc.
2. Students must handle the computers and other components carefully as they are expensive.
3. Before entering to lab must prepare for viva for which they are going to conduct experiment.
4. Before leaving the lab, check whether they have switched off the power supply and keep the chairs back properly.
5. Uniform and ID card are must.
6. Laboratory has to be kept clean and discipline has to be maintained.

SYLLABUS

PART-A: Simulation experiments using NS2/ NS3/ OPNET/ NCTUNS/ NetSim/QualNet or any other equivalent tool

1. Implement a point to point network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth.
2. Implement a four node point to point network with links n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP.
3. Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.
4. Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.
5. Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.
6. Implementation of Link state routing algorithm.

PART-B: Implement the following in C/C++

1. Write a program for a HLDC frame to perform the following.
   i) Bit stuffing
   ii) Character stuffing.
2. Write a program for distance vector algorithm to find suitable path for transmission.
3. Implement Dijkstra's algorithm to compute the shortest routing path.
4. For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases
   a. Without error
   b. With error
5. Implementation of Stop and Wait Protocol and Sliding Window Protocol
6. Write a program for congestion control using leaky bucket algorithm.

COURSE OUTCOMES

On the completion of this laboratory course, the students will be able to:

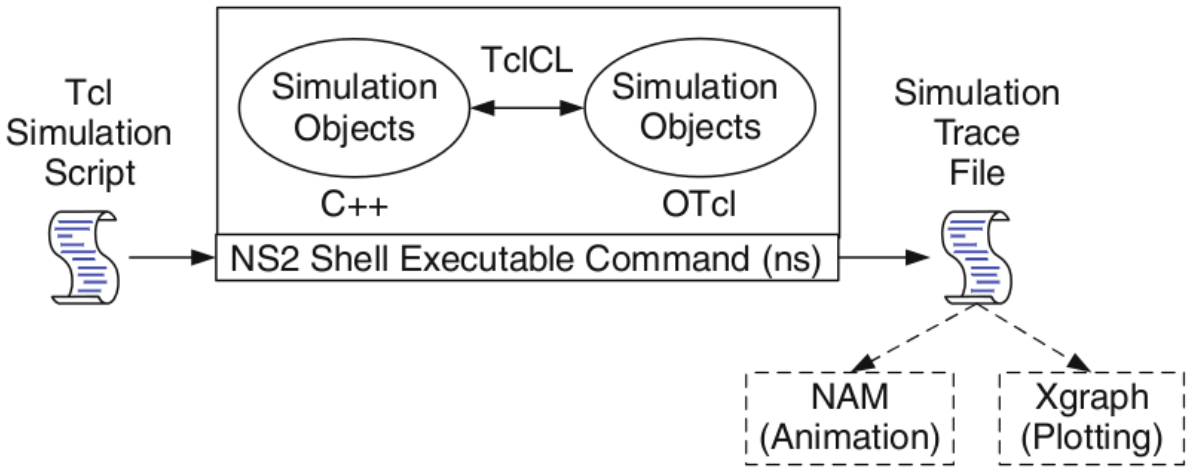
1. Use the network simulator for learning and practice of networking algorithms.
2. Illustrate the operations of network protocols and algorithms using C programming.
3. Simulate the network with different configurations to measure the performance parameters.
4. Implement the data link and routing protocols using C programming.

CONTENTS

INTRODUCTION TO NS 2

Use the network simulator for learning and practice of networking algorithms. Illustrate the operations of network protocols and algorithms using C programming. Simulate the network with different configurations to measure the performance parameters. Implement the data link and routing protocols using C programming.

Basic Architecture of NS2



Tcl scripting

- Tcl is a general purpose scripting language. [Interpreter]
- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL

Syntax: command arg1 arg2 arg3

☐ **Hello World!**

```
puts stdout{Hello, World!}
Hello, World!
```

☐ **Variables** Command Substitution

```
set a 5 set len [string length foobar]
```

set b $a set len [expr [string length foobar] + 9]

Simple Arithmetic
expr 7.2 / 4
☐ **Procedures**

proc Diag {a b} {
set c [expr sqrt($a * $a + $b * $b)]
return $c }
puts —Diagonal of a 3, 4 right triangle is [Diag 3 4]‖
Output: Diagonal of a 3, 4 right triangle is 5.0
☐ **Loops**

while{$i < $n} { for {set i 0} {$i < $n} {incr i} {
. . . . . .
}

Wired TCL Script Components
Create the event scheduler
Open new files & turn on the tracing
Create the nodes
Setup the links
Configure the traffic type (e.g., TCP, UDP, etc)
Set the time of traffic generation (e.g., CBR, FTP)
Terminate the simulation

NS Simulator Preliminaries.
1. Initialization and termination aspects of the ns simulator.

2. Definition of network nodes, links, queues and topology.

3. Definition of agents and of applications.

4. The nam visualization tool.

5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2
An ns simulation starts with the command

Set ns [new Simulator]

Which is thus the first line in the tcl script? This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.
In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using —open‖ command:

#Open the Trace file

```
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
```

#Open the NAM trace file

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

The above creates a trace file called —out.tr‖ and a nam visualization trace file called —out.nam‖. Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called —tracefile1‖ and —namfile‖ respectively. Remark that they begins with a # symbol. The second line open the file —out.tr‖ to be used for writing, declared with the letter —w‖. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command $ns flush-trace. In our case, this will be the file pointed at by the pointer —$namfile‖, i.e the file —out.tr‖.
The termination of the program is done using a —finish‖ procedure.

#Define a „finish“ procedure

```
Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close $namfile
Exec nam out.nam &
Exit 0
}
```

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method —**flush-trace"** will dump the traces on the respective files. The tcl command —**close"** closes the trace files defined before and **exec** executes the nam program for visualization. The command **exit** will ends the application and return the number 0 as status to the system. Zero is the default for a clean exit. Other values can be used to say that is a exit because something fails.

At the end of ns program we should call the procedure —finish‖ and specify at what time the termination should occur. For example,

```
$ns at 125.0 "finish"
```

will be used to call —**finish**‖ at time 125sec.Indeed,the **at** method of the simulator allows us to schedule events explicitly.
The simulation can then begin using the command

```
$ns run
```

Definition of a network of links and nodes
The way to define a node is

```
set n0 [$ns node]
```

The node is created which is printed by the variable n0. When we shall refer to that node in the script we shall thus write $n0.
Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

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

$ns at 125.0 "finish"
Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.
To define a directional link instead of a bi-directional one, we should replace —duplex-link‖ by —simplex-link‖.

In NS, an output queue of a node is implemented as a part of each link whose input is that node. The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Discard) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).

In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

#set Queue Size of link (n0-n2) to 20
$ns queue-limit $n0 $n2 20

Agents and Applications
We need to define routing (sources, destinations) the agents (protocols) the application that use them.

FTP over TCP
TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

set tcp [new Agent/TCP]

The command **$ns attach-agent $n0 $tcp** defines the source node of the tcp connection.
The command

set sink [new Agent /TCPSink]

Defines the behavior of the destination node of TCP and assigns to it a pointer called sink.

#Setup a UDP connection

set udp [new Agent/UDP]

$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2

#setup a CBR over UDP connection

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false

Above shows the definition of a CBR application using a UDP agent
The command **$ns attach-agent $n4 $sink** defines the destination node. The command **$ns connect $tcp $sink** finally makes the TCP connection between the source and destination nodes.
TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command **$tcp set packetSize_ 552**.
When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **$tcp set fid_ 1** that assigns to the TCP connection a flow identification of —1‖.We shall later give the flow identification of —2‖ to the UDP connection.

CBR over UDP

A UDP source and destination is defined in a similar way as in the case of TCP. Instead of defining the rate in the command $cbr set rate_ 0.01Mb, one can define the time interval between transmission of packets using the command.

$cbr set interval_ 0.005

The packet size can be set to some value using

$cbr set packetSize_ <packet size>

Scheduling Events

NS is a discrete event based simulation. The tcp script defines when event should occur. The initializing command set ns [new Simulator] creates an event scheduler, and events are then scheduled using the format:

$ns at <time> <event>

The scheduler is started when running ns that is through the command $ns run. The beginning and end of the FTP and CBR application can be done through the following command

$ns at 0.1 "$cbr start"
$ns at 1.0 " $ftp start"
$ns at 124.0 "$ftp stop"
$ns at 124.5 "$cbr stop"

Structure of Trace Files

When tracing into an output ASCII file, the trace is organized in 12 fields as follows in fig shown below, The meaning of the fields are:

Event	Time	From Node	To Node	PKT Type	PKT size	Flags	Fid	Src Address	Dst Address	Seq No	Pkt ID

1. The first field is the event type. It is given by one of four possible symbols r, +, -, d which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.

2. The second field gives the time at which the event occurs.

3. Gives the input node of the link at which the event occurs.

4. Gives the output node of the link at which the event occurs.

5. Gives the packet type (eg CBR or TCP)

6. Gives the packet size

7. Some flags

8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.

9. This is the source address given in the form of —node.portǁ.

10. This is the destination address, given in the same form.

11. This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes

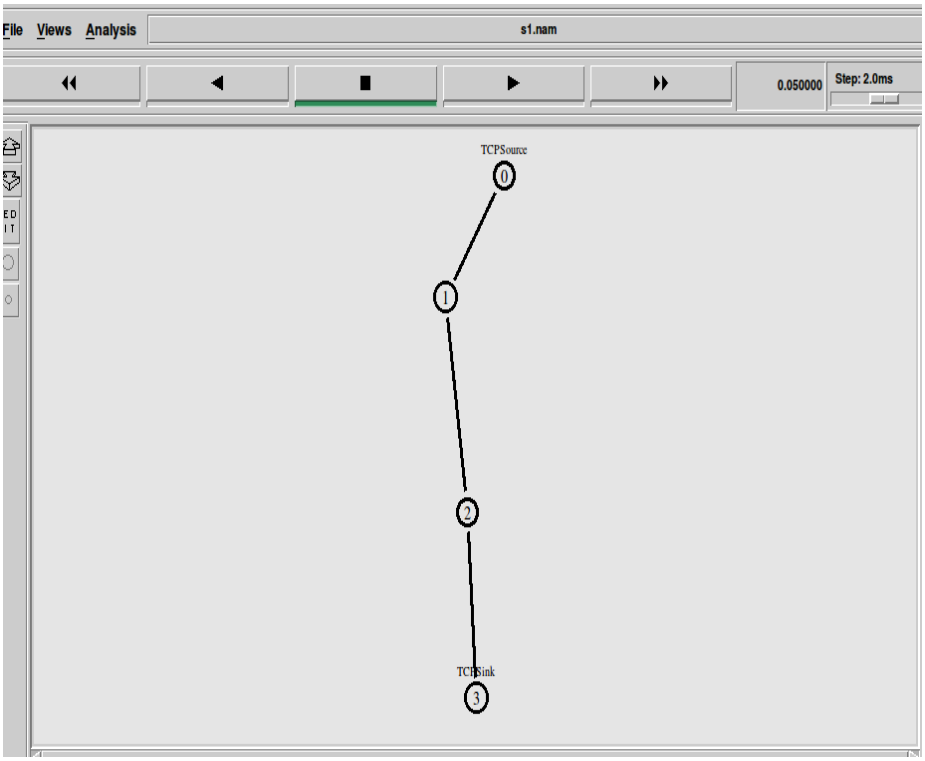
12. The last field shows the Unique id of the packet.

PART-A

Experiment No. 1

Four-Node Point-to-Point Network

Aim: Implement a point-to-point network with duplex links between them. Analyze the network performance by setting the queue size and vary the bandwidth and find the number of packets dropped.



s1.awk

```
BEGIN{
        count = 0;
}
{
        event = $1;
        if(event == "d")   { count++; }
}

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

s1.tcl

```
#create new simulation instance
set ns [new Simulator]

#open trace file
set tracefile [open s1.tr w]
```

```
$ns trace-all $tracefile

#open nam:animation file
set namfile [open s1.nam w]
$ns namtrace-all $namfile

#define finish procedure to perform at the end of simulation
proc finish {} {
        global ns tracefile namfile
        $ns flush-trace

        #dump all traces and close files
        close $tracefile
        close $namfile

        #execute nam animation file
        exec nam s1.nam &

        #execute awk file in background
        exec awk -f s1.awk s1.tr &
        exit 0
}

#create 4 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#create labels
$n0 label "TCPSource"
$n3 label "TCPSink"

#set color
$ns color 1 red

#create link between nodes /create topology
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail

#set queue size of N packets between n2 and n3
$ns queue-limit $n2 $n3 5

#create TCP agent and attach to node 0
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
```

```
#create TCP sink agent and attach to node 3
set tcpsink [new Agent/TCPSink]
$ns attach-agent $n3 $tcpsink

#create traffic: FTP: create FTP source agent on top of TCP and attach to TCP agent
set ftp [new Application/FTP]
$ftp attach-agent $tcp

#connect TCP agent with TCP sink agent
$ns connect $tcp $tcpsink

#set the color
$tcp set class_  1

#schedule events
$ns at 0.2 "$ftp start"
$ns at 2.5 "$ftp stop"
$ns at 3.0 "finish"
$ns run
```

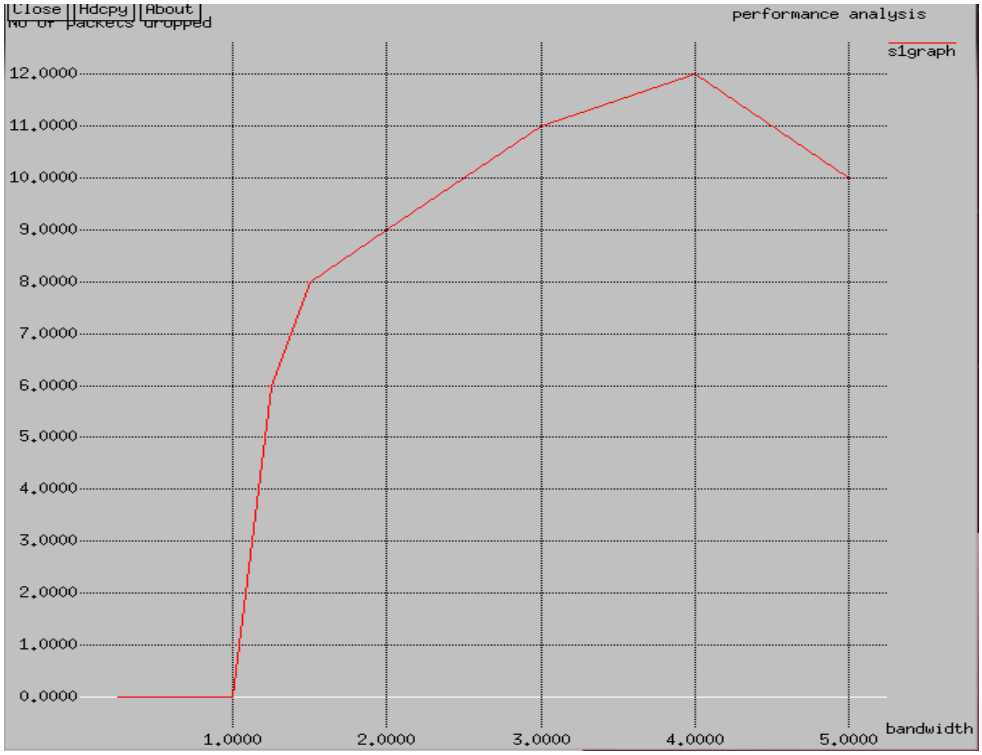
Steps

1) In the command line type xed s1.awk
2) xed s1.tcl
3) to run ns s1.tcl and ns s1.awk
4) run awk –f s1.awk s1.tr
5) Vary the bandwidth from node 0 to 2 (0.25 to 5Mb) and keeping same bandwidth (1Mb) between 2 to 3 and note down packets dropped.
6) Xed s1graph and type the above result as below

s1graph

```
0.25  0
0.50  0
0.75  0
1.00  0
1.25  6
1.50  8
2.00  9
3.00 11
4.00 12
5.00  10
```

7) xgraph -x "Bandwidth(Mbps)" -y "No of packets dropped" -t "Performance analysis" **s1graph**



Result:

Exercise:

1) Implement a point – to – point network with three nodes (for given orientation) with duplex links between them. Analyze the network performance by setting the queue size and vary the bandwidth and find the number of packets dropped using UDP application

2) Implement a ring topology network with four nodes with duplex links between them. Set colors to the nodes, change the buffer type and use CBR traffic

Trace Format Example

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive (at to_node)
+ : enqueue (at queue)              src_addr : node.port (3.0)
- : dequeue (at queue)              dst_addr : node.port (0.0)
d : drop    (at queue)
```

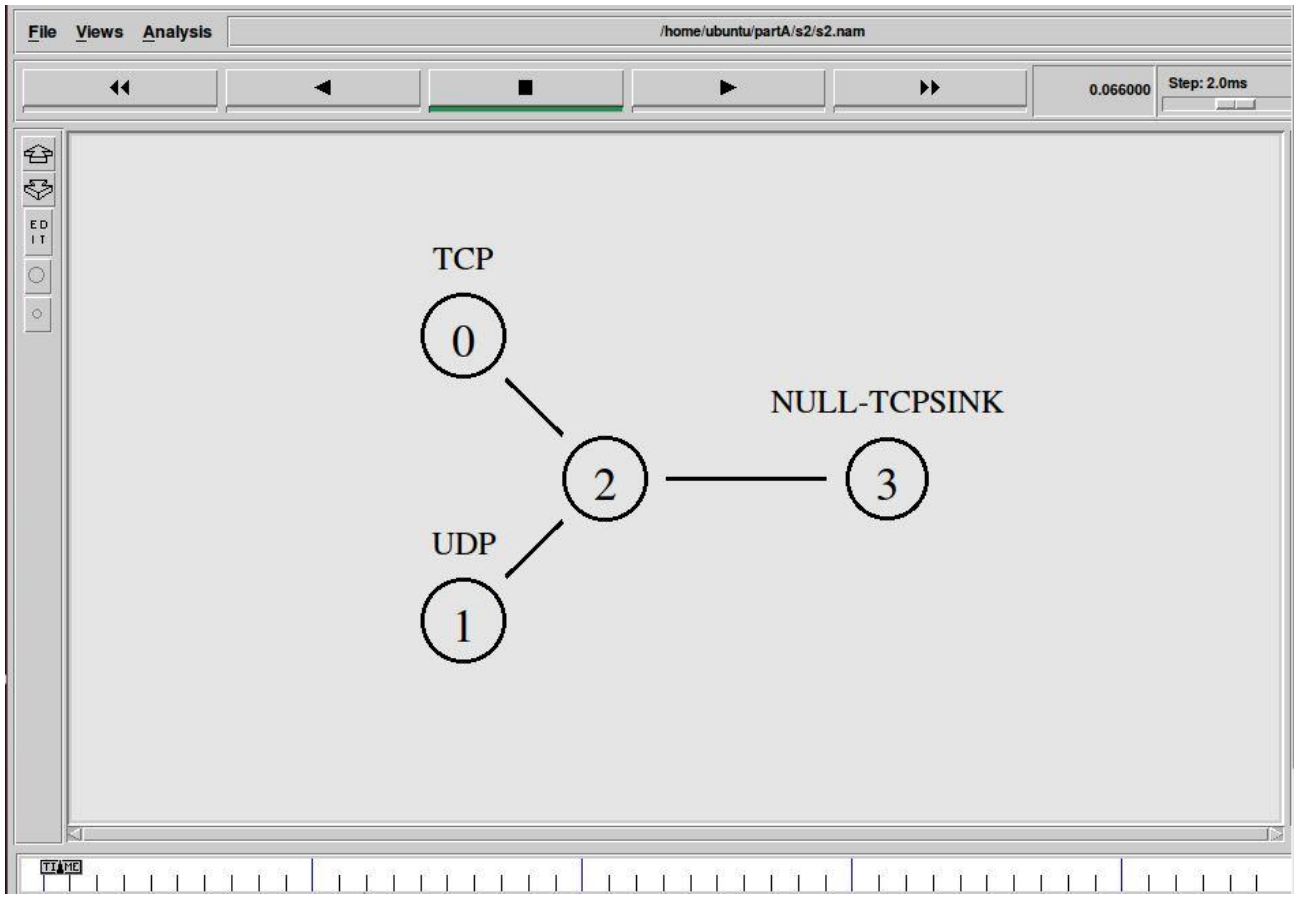
```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

1. An event (+, -, d, r) descriptor
2. Simulation time (in seconds) of that event
3. From node
4. To node (3, 4 identifies the link on which the event occurred)
5. Packet type (in Bytes)
6. Packet size (in Bytes)
7. Flags (appeared as "------" since no flag is set)
8. Flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script. Even though fid field may not be used in a simulation, users can use this field for analysis purposes. The fid field is also used when specifying stream color for the NAM display.
9. Source address in forms of "**node.port**".
10. Destination address in forms of "**node.port**".
11. Network layer protocol's packet sequence number. Note that even though UDP implementations do not use sequence number, NS keeps track of UDP packet sequence number for analysis purposes.
12. Unique id of the packet

Experiment No. 2

Point-to-Point Network for the given Application

Aim: To implement a four node point-to-point network with the links connected as follows: n0–n2, n1–n2 and n2–n3. Apply TCP agent between n0–n3 and UDP between n1–n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP / UDP



s2.awk

```
BEGIN {
        ctcp = 0;
        cudp = 0;
}
{
        pkt = $5;
        if(pkt == "cbr") { cudp++; }
        if(pkt == "tcp") { ctcp++; }
}
END {
        printf("\nNo of packets sent\nTcp : %d \n Udp : %d\n", ctcp, cudp);
}
```

s2.tcl

```
set ns [new Simulator]

set namfile [open s2.nam w]
$ns namtrace-all $namfile
set tracefile [open s2.tr w]
$ns trace-all $tracefile

proc finish {} {
        global ns namfile tracefile
        $ns flush-trace
        close $namfile
        close $tracefile

        exec nam s2.nam &
        exec awk -f s2.awk s2.tr &
        exit 0
}

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

$n0 label "TCP"
$n1 label "UDP"
$n3 label "NULL-TCPSINK"

$ns color 1 red
$ns color 2 blue

$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 2.75Mb 20ms DropTail

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set tcpsink [new Agent/TCPSink]
$ns attach-agent $n3 $tcpsink

set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

```
$ns connect $tcp $tcpsink

#Create a UDP Agent and attach to the node n1
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp

#Create a Null Agent and attach to the node n3
set null [new Agent/Null]
$ns attach-agent $n3 $null

#Create a CBR Traffic source and attach to the UDP Agent
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp

#Specify the Packet Size and interval
$cbr set packetSize_ 500
$cbr set interval_ 0.005

#Connect the CBR Traffic source to the Null agent
$ns connect $udp $null


$tcp set class_ 1
$udp set class_ 2

$ns at 0.2 "$cbr start"
$ns at 0.1 "$ftp start"
$ns at 4.5 "$cbr stop"
$ns at 4.4 "$ftp stop"
$ns at 5.0 "finish"
$ns run
```

Output

xed s2.awk
xed s2.tcl
ns s2.tcl
ns s2.awk
awk –f s2.awk s2.tr
Vary bandwidth and note down results

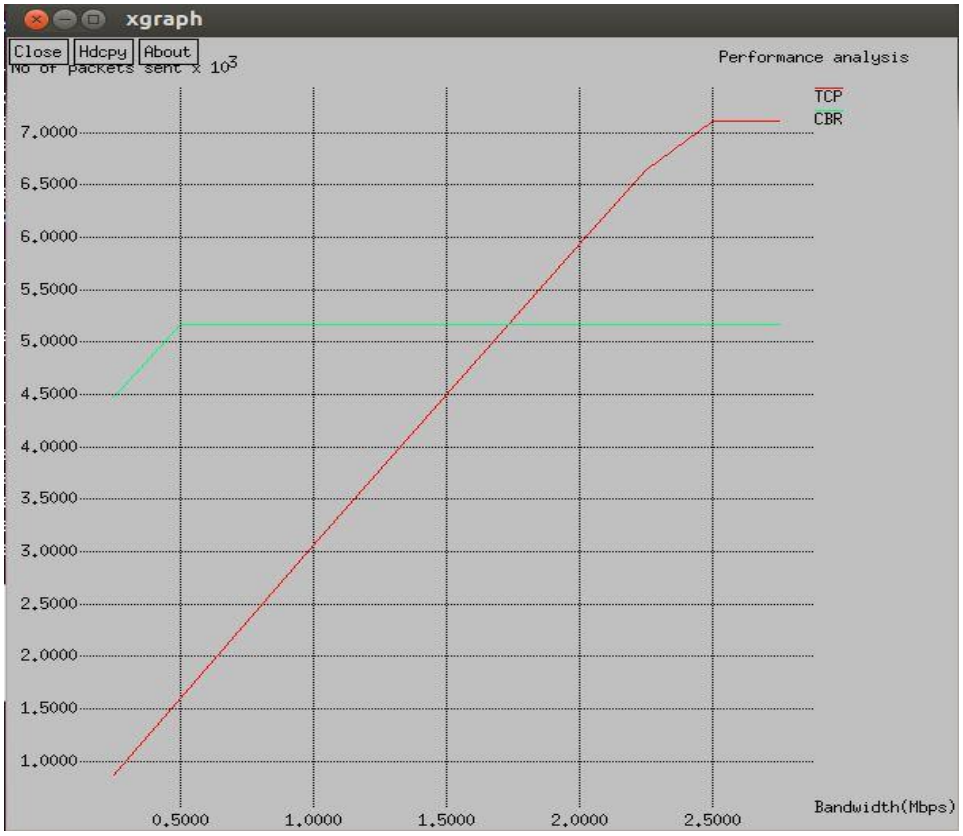
No of packets sent
Tcp : 5940
Udp : 5166

xed s2graph

s2graph

```
"TCP"
0.25    869
0.50    1608
0.75    2334
1.00    3066
1.25    3780
1.50    4500
1.75    5220
2.00    5940
2.25    6642
2.50    7110
2.75    7110

"CBR"
0.25    4478
0.50    5166
0.75    5166
1.00    5166
1.25    5166
1.50    5166
1.75    5166
2.00    5166
2.25    5166
2.50    5166
2.75    5166
```

#xgraph -x "Bandwidth(Mbps)" -y "No of packets sent" -t "Performance analysis" **s2graph**

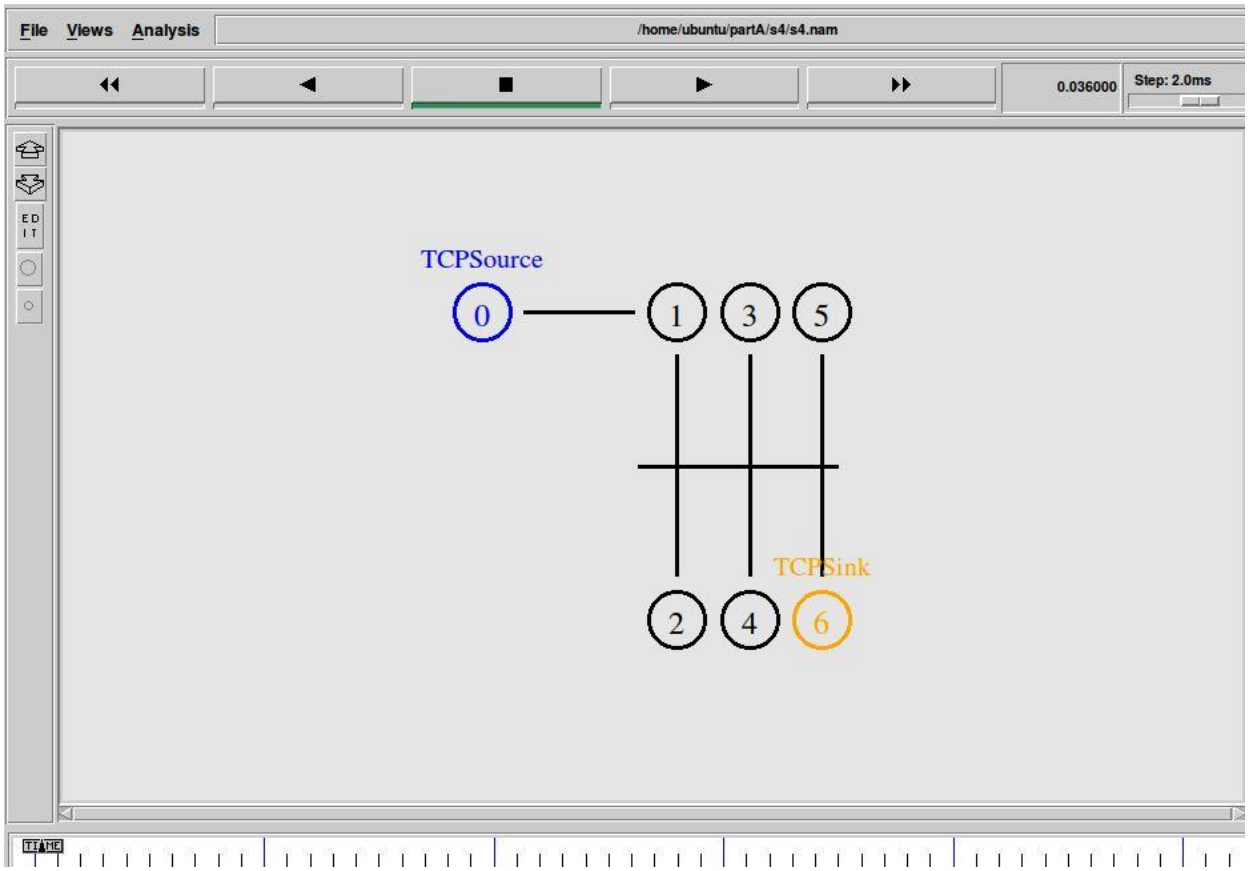


Result:

Experiment No. 3

Ethernet LAN using N Nodes

Aim: To implement Ethernet LAN using N nodes (6-10), change error rate and data rate, and compare throughput.



s3.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 {
}
```

s3.tcl

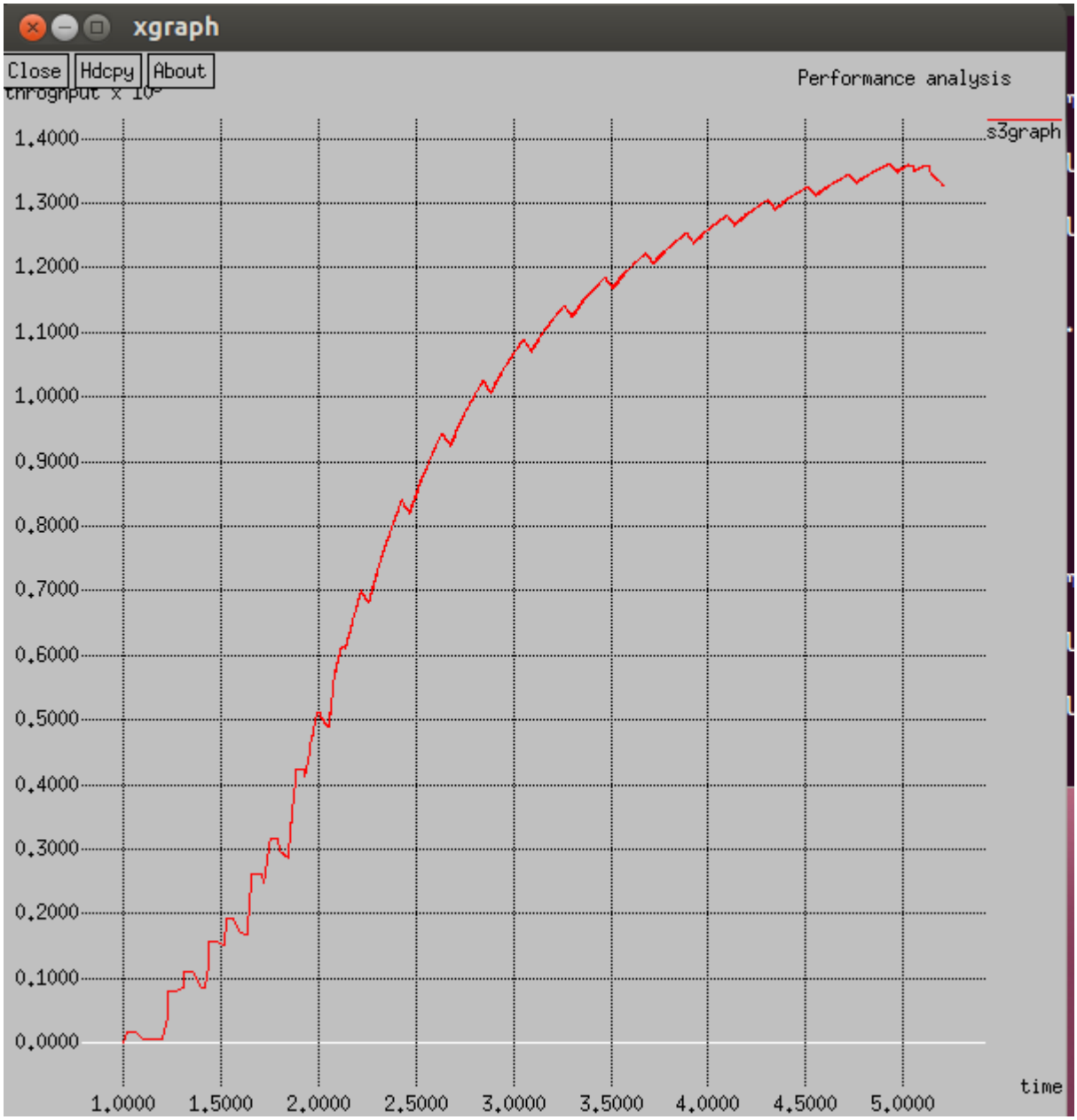
```
set ns [new Simulator]

set namfile [open s3.nam w]
$ns namtrace-all $namfile

set tracefile [open s3.tr w]
$ns trace-all $tracefile

proc finish {} {
        global ns namfile tracefile
        $ns flush-trace
        close $namfile
        close $tracefile
        exec nam s3.nam &
        exec awk -f s3.awk s3.tr  >  s3graph &
        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]

$n0 label "TCPSource"
```

```
$n6 label "TCPSink"

$ns color 1 red

$n0 color blue
$n6 color orange

$ns duplex-link $n0 $n1 3Mb 20ms DropTail
$ns make-lan "$n1 $n2 $n3 $n4 $n5 $n6" 2Mb 40ms LL Queue/DropTail Mac/802_3

$ns duplex-link-op $n0 $n1 orient right

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

$tcp set class_ 1

set tcpsink [new Agent/TCPSink]
$ns attach-agent $n6 $tcpsink

set ftp [new Application/FTP]
$ftp attach-agent $tcp

$ns connect $tcp $tcpsink

$ns at 1.0 "$ftp start"
$ns at 5.0 "$ftp stop"
$ns at 5.5 "finish"
$ns run
```

Output:

#xed s3.tcl
#ns s3.tcl

#xgraph –x “Time” –y “Throughput” –t “Performance analysis” **s3graph**

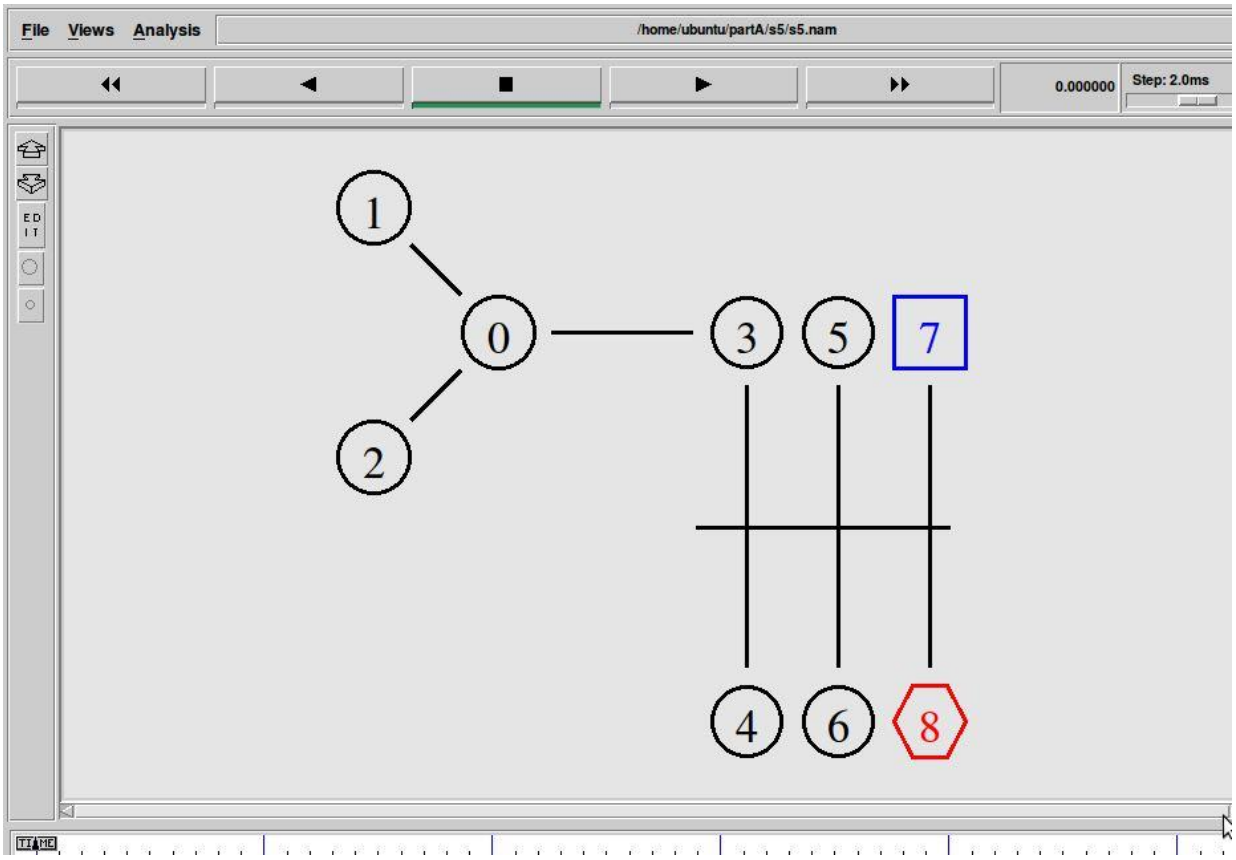

xgraph
Close
Hdcpy
About
Performance analysis
throughput x 10^6
s3graph
1.4000
1.3000
1.2000
1.1000
1.0000
0.9000
0.8000
0.7000
0.6000
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
1.0000
1.5000
2.0000
2.5000
3.0000
3.5000
4.0000
4.5000
5.0000
time

Result:

Experiment No. 4

Ethernet LAN for Multiple Traffic

Aim: To simulate an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.



s4.awk

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

s4.tcl

```
        set ns [new Simulator]

        set namfile [open s4.nam w]
        $ns namtrace-all $namfile

        set tracefile [open s4.tr w]
        $ns trace-all $tracefile

        proc finish {} {
                global ns namfile tracefile
                $ns flush-trace
                close $namfile
                close $tracefile
                exec nam s4.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]

        $ns color 1 Blue
        $ns color 2 Red

        $n7 shape box
        $n7 color Blue
        $n8 shape hexagon
        $n8 color Red

        $ns duplex-link $n1 $n0 2Mb 10ms DropTail
        $ns duplex-link $n2 $n0 2Mb 10ms DropTail
        $ns duplex-link $n0 $n3 1Mb 20ms DropTail

$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 40ms LL Queue/DropTail Mac/802_3

        $ns duplex-link-op $n1 $n0 orient right-down
        $ns duplex-link-op $n2 $n0 orient right-up
        $ns duplex-link-op $n0 $n3 orient right

        $ns queue-limit $n0 $n3 20

        set tcp1 [new Agent/TCP/Vegas]
```

```
$ns attach-agent $n1 $tcp1

set sink1 [new Agent/TCPSink]
$ns attach-agent $n7 $sink1

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

$ns connect $tcp1 $sink1

$tcp1 set class_ 1
$tcp1 set packetSize_ 55

set tfile1 [open cwnd1.tr w]
$tcp1 attach $tfile1
$tcp1 trace cwnd_

set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $n2 $tcp2

set sink2 [new Agent/TCPSink]
$ns attach-agent $n8 $sink2

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

$ns connect $tcp2 $sink2

$tcp2 set class_ 2
$tcp2 set packetSize_ 55

set tfile2 [open cwnd2.tr w]
$tcp2 attach $tfile2
$tcp2 trace cwnd_

$ns at 0.5 "$ftp1 start"
$ns at 1.0 "$ftp2 start"
$ns at 5.0 "$ftp2 stop"
$ns at 5.0 "$ftp1 stop"
$ns at 5.5 "finish"
$ns run
```

Output

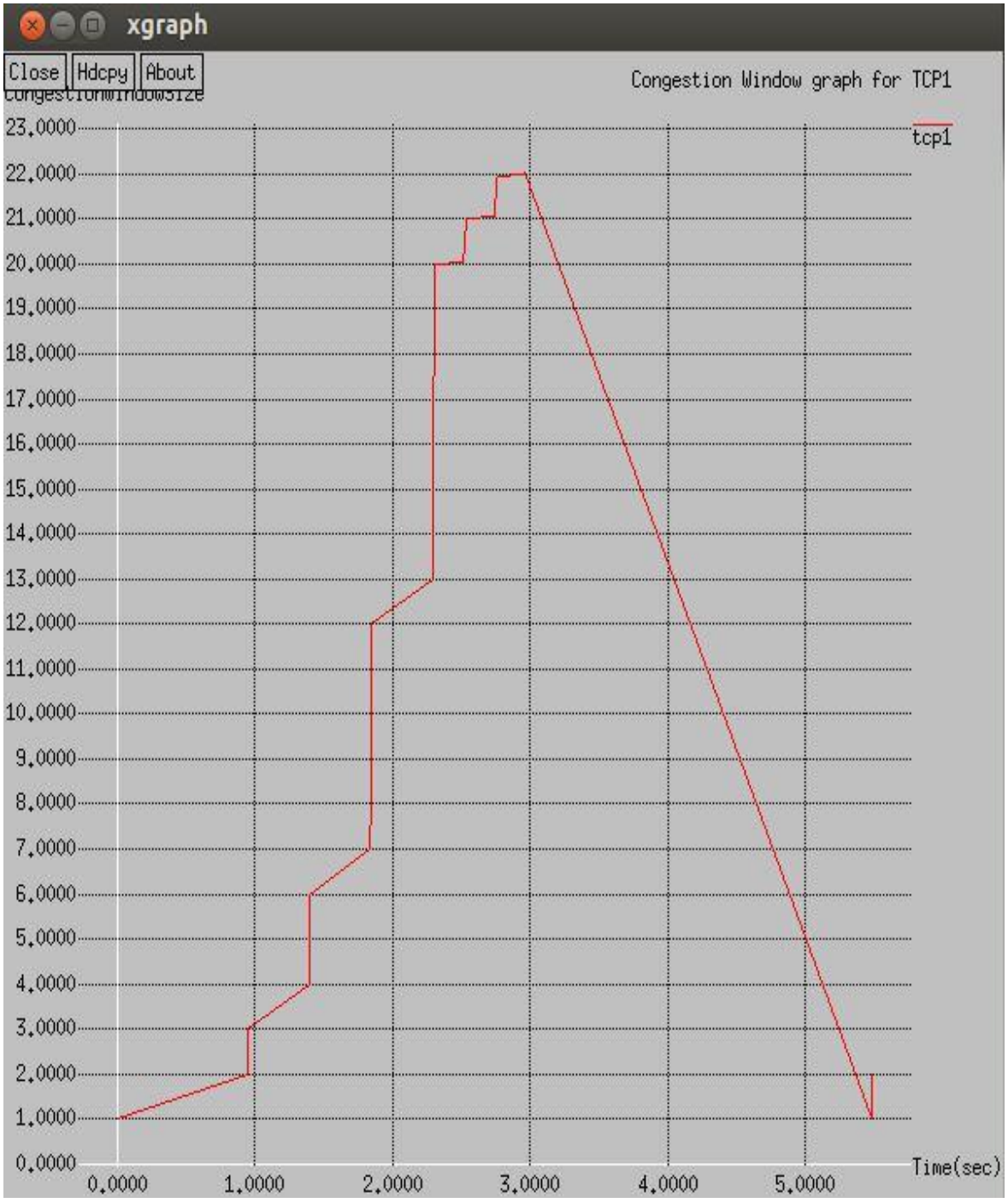
```
#xed s4.tcl
#  ns s4.tcl
```

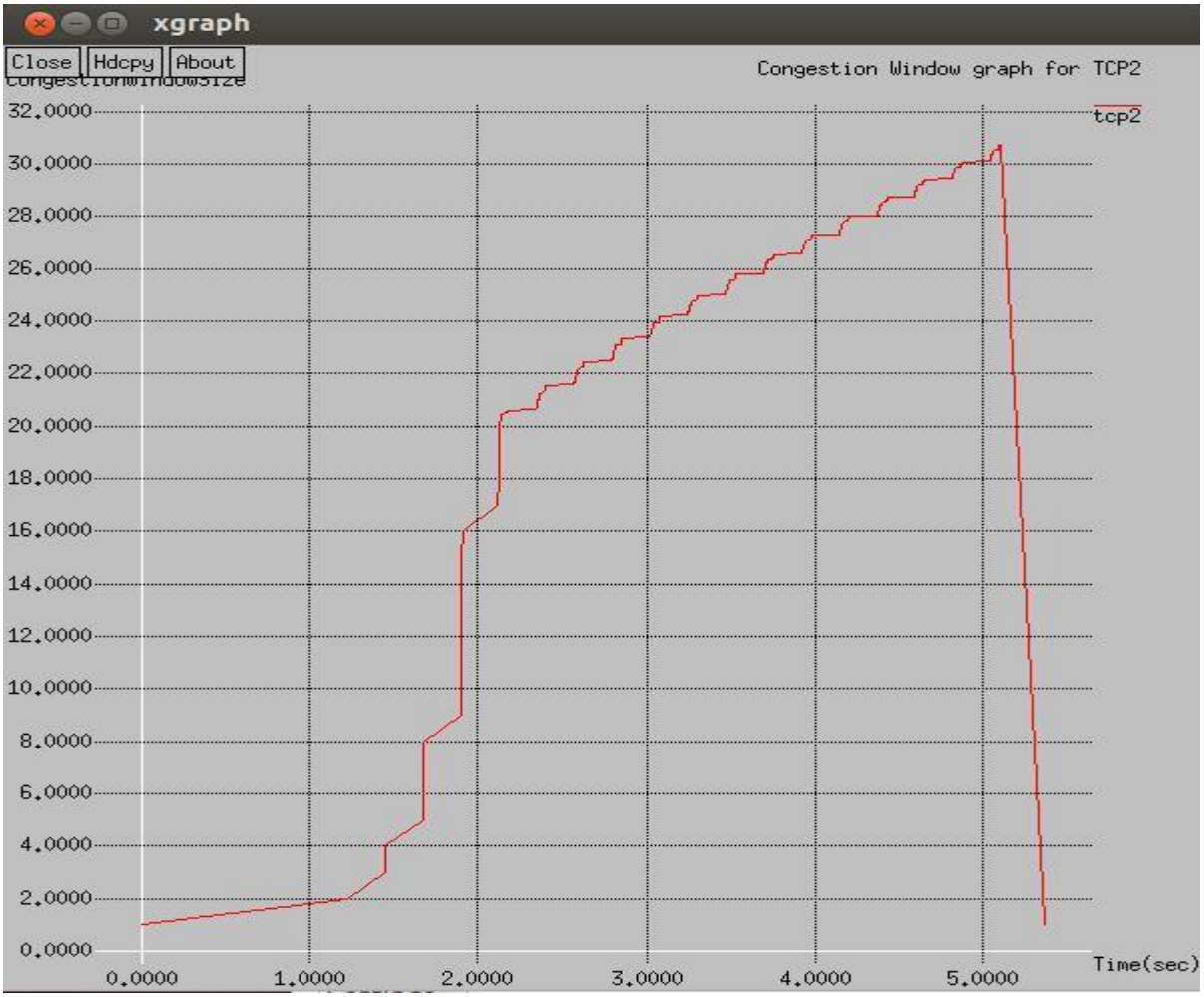
exec awk -f s4.awk cwnd1.tr > tcp1 &

#xgraph -x "Time(sec)" -y "CongestionWindowSize" -t "Congestion Window graph for TCP1" **tcp1**

exec awk -f s4.awk cwnd2.tr > tcp2 &

#xgraph -x "Time(sec)" -y "CongestionWindowSize" -t "Congestion Window graph for TCP2" **tcp2**





Exercise: Implement Ethernet LAN using n nodes and assign multiple traffic nodes and plot congestion window for different source / destination.

cwnd trace file example

0.00000	**2**	**0**	**1**	**0**	**cwnd_**	**1.000**
(1)	(2)	(3)	(4)	(5)	(6)	(7)

1. Timestamp
2. Source node id of the flow
3. Source port id
4. Destination node id of the flow
5. Destination port id

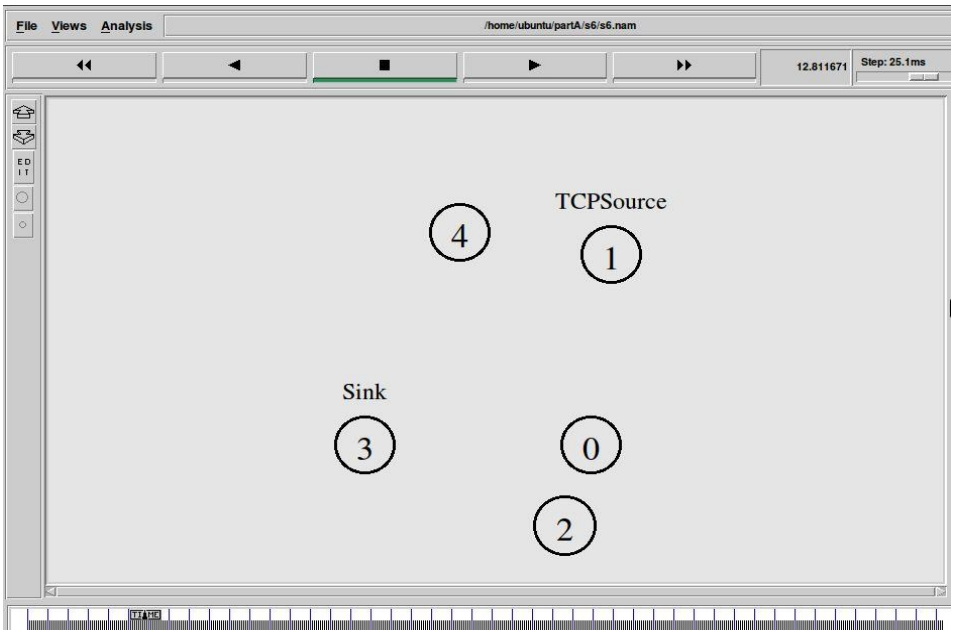
6. Name of the variable being traced (cwnd_ , t_seqno_ , throughput, reverse_feedback)
7. Value of the traced variable

Result

Experiment No. 5

Wireless LAN

Aim: To simulate simple ESS and with transmitting nodes in wireless LAN by simulation and determine the performance with respect to transmission of packets.



s5.awk

```
BEGIN{
        PacketRcvd = 0;
        Throughput = 0.0;
}
{
        if(($1 == "r")&&($3 == "_3_")&&($4 == "AGT")&&($7 == "tcp")&&($8 > 1000))
        {
                PacketRcvd++;
        }
}
END {
        Throughput = ((PacketRcvd*1000*8) / (95.0*1000000));
        printf("\nThe throughput is:%f\n", Throughput);
}
```

s6.tcl

```
set ns [new Simulator]
set tracefile [open s5.tr w]
```

```
$ns trace-all $tracefile

set namfile [open s5.nam w]
$ns namtrace-all-wireless $namfile 750 750

proc finish {} {
        global ns tracefile namfile
        $ns flush-trace
        close $tracefile
        close $namfile
        exec nam s5.nam &
        exec awk -f s5.awk s5.tr &
        exit 0
}

#get the number of nodes value from the user
set  val(nn)   5
#create new topography object
set topo [new Topography]
$topo load_flatgrid 750 750

#Configure the nodes
$ns     node-config    -adhocRouting     AODV \
-llType          LL \
-macType      Mac/802_11 \
-ifqType        Queue/DropTail \
-channelType Channel/WirelessChannel \
-propType      Propagation/TwoRayGround \
-antType       Antenna/OmniAntenna \
-ifqLen          50 \
-phyType       Phy/WirelessPhy \
-topoInstance          $topo \
-agentTrace   ON \
-routerTrace  ON \
-macTrace     OFF \
-movementTrace ON

#general operational descriptor storing the hop details in the network
set god_ [create-god $val(nn)]

#create mobile nodes
for {set i 0} {$i < $val(nn)} {incr i} {
        set n($i) [$ns node]
}

#label node
$n(1) label "TCPSource"
$n(3) label "Sink"
```

```
#Randomly placing the nodes
for {set i 0} {$i < $val(nn)} {incr i} {
        set XX [expr rand()*750]
        set YY [expr rand()*750]
        $n($i) set X_ $XX
        $n($i) set Y_ $YY
}

#define the initial position for the nodes
for {set i 0} {$i < $val(nn)} {incr i} {
        $ns initial_node_pos $n($i) 100
}

#define the destination procedure to set the destination to each node
proc destination {} {
        global ns val n
        set now [$ns now]
        set time 5.0
        for {set i 0} {$i < $val(nn)} {incr i} {
                set XX [expr rand()*750]
                set YY [expr rand()*750]
                $ns at [expr $now + $time] "$n($i) setdest $XX $YY 20.0"
        }
        $ns at [expr $now + $time] "destination"
}
set tcp [new Agent/TCP]
$ns attach-agent $n(1) $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n(3) $sink

$ns connect $tcp $sink

$ns at 0.0 "destination"
$ns at 1.0 "$ftp start"
$ns at 100 "finish"
$ns run
```

Output

#xed s5.tcl

ns s5.tcl **5**

The throughput is: 0.579368

Experiment No. 6

Link State Routing Algorithm

Aim: To implement Link state routing algorithm

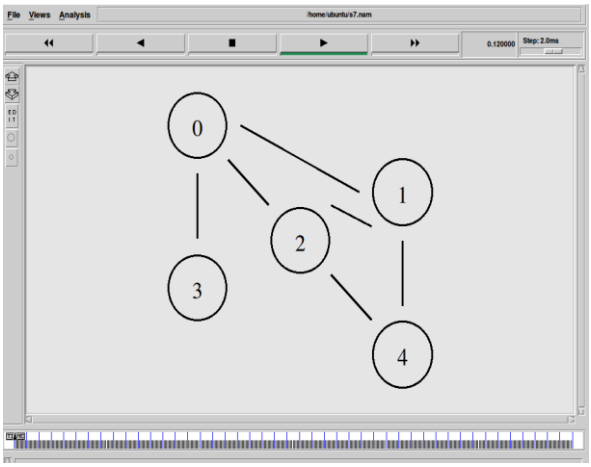
```
set ns [new Simulator]
set namfile [open s6.nam w]
$ns namtrace-all $namfile

set tracefile [open s6.tr w]
$ns trace-all $tracefile

proc finish {} {
        global ns namfile tracefile
        $ns flush-trace
        close $namfile
        close $tracefile
        exec nam s6.nam &
        exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n4 1Mb 10ms DropTail
$ns duplex-link $n2 $n4 1Mb 10ms DropTail

$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n0 $n3 orient down
$ns duplex-link-op $n1 $n2 orient left-down
$ns duplex-link-op $n1 $n4 orient down
$ns duplex-link-op $n2 $n4 orient right-down

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $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 $n4 $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n2 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set  interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n4 $null0
$ns connect $udp1 $null0

$ns rtproto LS

$ns rtmodel-at 2.0 down $n1 $n4
$ns rtmodel-at 2.5 up $n1 $n4
$ns rtmodel-at 3.0 down $n2 $n4
$ns rtmodel-at 3.5 up $n2 $n4
$udp0 set class_ 1
$udp1 set class_ 2
$ns color 1 Red
$ns color 2 Green

$ns at 1.0 "$cbr0 start"
$ns at 1.5 "$cbr1 start"

$ns at 10 "finish"
        $ns run
```

Output

#xed s6.tcl

ns s6.tcl



Part-B

Experiment No. 1

Bit and Character Stuffing

Aim: To write a program for a HDLC frame to perform the following i) Bit stuffing ii) Character stuffing

i) Bit stuffing

```
#include<stdio.h>
#include<string.h>
#define Max 100
void main()
 {
int si=0,di, count=6;
char flag []="01111110", src[MAX],dest[Max];
printf("\n\n ----------------bit stuffing----------------------\n\n");
printf("enter the data in binary:\t");
scanf("%s",src);
strcpy(dest,flag);
di=strlen(flag);
while (src[si]!='\0')
{
If (src[si]=='1')
count++;
else
count=0;
dest[di]=src[si];
di++;
si++;
if(count==5)
{
dest[di]='0';
di++;
count=6;
}
}
dest[di]='\0';
strcat(dest,flag);
printf("\n\n stuffed data is : \t %s ", dest);
}
```

Output:

----------bit stuffing----------

Enter the data in binary : 1110111111011

Stuffed data is : **0111110**1110111110101**0111110**

ii) Character stuffing

```
#include<stdio.h>
#include<string.h>
#define max 100
int main()
 {
int si=0,di=0;
char begin[]="*", end[]="*";
char src[max],dest[max];
printf("\n\n ----------character stuffing----------\n\n");
printf("enter the frame data in ASCII:\t");
gets(src);
strcpy(dest,begin);
di=strlen(begin);
while(src[si]!='\0')
{
if (src[si]=='*')
{

dest[di]='$',dest[di+1]='*';
si=si+1;
di=di+2;
}
else if(src[si]=='$')
{
dest[di]='$', dest[di+1]='$';
si=si+1;
di=di+2;
}
else dest[di++]=src[si++];
}
dest[di]='\0';
strcat(dest,end);
printf("\n\n character stuffed frame is : \t %s ", dest);
getchar();
}
```

Output:

----------character stuffing----------

enter the frame data in ASCII:CCN*LAB$EXAM

character stuffed frame is: *CCN**$**LAB**$**$EXAM***

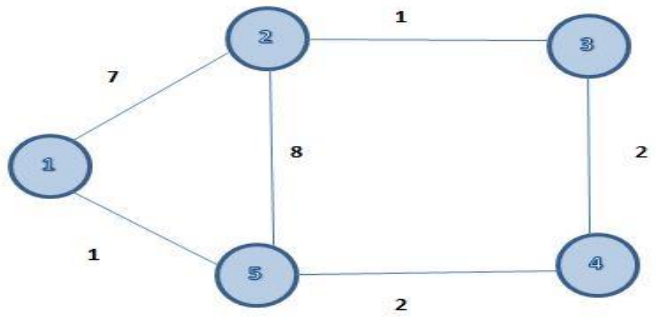
Experiment No. 2

Distance Vector Algorithm

Aim: To write a program for distance vector algorithm to find suitable path for transmission.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
	int d[10][10], t[10][10], a[10][10], i, j, n, k, count;
	printf("Enter the number of nodes\n");
	scanf("%d", &n);
	printf("Enter the initial cost matrix\n");
	for(i=1; i<=n; i++)
	{
		for(j=1; j<=n; j++)
		{
				scanf("%d", &d[i][j]);
				d[i][i] = 0;
				t[i][j] = d[i][j];
				a[i][j] = j;
		}
	}
	do
	{
		count = 0;
		for(i=1; i<=n; i++)
		{
				for(j=1; j<=n; j++)
				{
				for(k=1; k<=n; k++)
				{
					if(t[i][j]>(d[i][k]+t[k][j]))
				{
					t[i][j] = (t[i][k] + t[k][j]);
					a[i][j] = k;
					count++;
				}
				}
				}
		}
	}while(count!=0);
```

```
for(i=1; i<=n; i++)
{
                printf("\n\n For router %d\n", i);
                for(j=1; j<=n; j++)
                {
                printf("\t\nto node %d, via %d, Distance -> %d ",j, a[i][j], t[i][j]);
        }
        }
        printf("\n\n");
        return 0;
}
```



Initially:

	Node 1	Node 2	Node 3	Node 4	Node 5
Node 1	0	7	999	999	1
Node 2	7	0	1	999	8
Node 3	999	1	0	2	999
Node 4	999	999	2	0	2
Node 5	1	8	999	2	0

Finally:

	Node 1	Node 2	Node 3	Node 4	Node 5

Node 1	**0**	**6**	**5**	**3**	**1**
Node 2	**6**	**0**	**1**	**3**	**5**
Node 3	**5**	**1**	**0**	**2**	**4**
Node 4	**3**	**3**	**2**	**0**	**2**
Node 5	**1**	**5**	**4**	**2**	**0**

Output:

Enter the number of nodes : **5**

Enter the initial cost matrix :

0	**7**	**999**	**999**	**1**
7	**0**	**1**	**999**	**8**
999	**1**	**0**	**2**	**999**
999	**999**	**2**	**0**	**2**
1	**8**	**999**	**2**	**0**

For router 1

to node 1, via 1, Distance -> 0
to node 2, via 5, Distance -> 6
to node 3, via 5, Distance -> 5
to node 4, via 5, Distance -> 3
to node 5, via 5, Distance -> 1

For router 2

to node 1, via 3, Distance -> 6
to node 2, via 2, Distance -> 0
to node 3, via 3, Distance -> 1
to node 4, via 3, Distance -> 3
to node 5, via 3, Distance -> 5

For router 3

to node 1, via 4, Distance -> 5
to node 2, via 2, Distance -> 1
to node 3, via 3, Distance -> 0
to node 4, via 4, Distance -> 2
to node 5, via 4, Distance -> 4

For router 4

to node 1, via 5, Distance -> 3
to node 2, via 3, Distance -> 3
to node 3, via 3, Distance -> 2
to node 4, via 4, Distance -> 0
to node 5, via 5, Distance -> 2

For router 5

to node 1, via 1, Distance -> 1
to node 2, via 4, Distance -> 5
to node 3, via 4, Distance -> 4
to node 4, via 4, Distance -> 2
to node 5, via 5, Distance -> 0

Experiment No. 3

Dijkstra's Algorithm

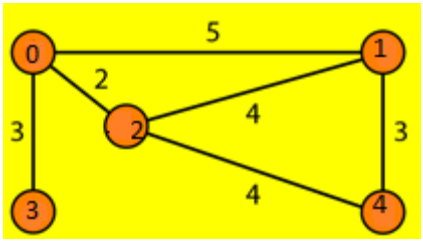
Aim: To implement Dijkstra's algorithm to compute the shortest routing path

```
#include<stdio.h>
#include<conio.h>
void dijkstra(int n, int v, int cost[10][10],int dist[10])
{
int count, u, i, w, visited[10], min;
for(i=0;i<n;i++)
{
visited[i]=0;
dist[i]=cost[v][i];
}
visited[v]=1;
dist[v]=0;
count=2;
while(count<=n)
{
min=999;
for(w=0;w<n;w++)
if((dist[w]<min) && (visited[w]!=1))
{
min=dist[w];
u=w;
}
visited[u]=1;
count++;
for(w=0;w<n;w++)
if((dist[u]+cost[u][w]<dist[w]) && (visited[w]!=1))
dist[w]=dist[u]+cost[u][w];
}
}
void main()
{
int n, v, cost[10][10], dist[10], i, j;
printf("Enter number of vertices:");
scanf("%d",&n);
printf("\nEnter cost matrix (for infinity, enter 999):\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&cost[i][j]);
printf("\nEnter source vertex:");
scanf("%d",&v);
```

```
dijkstra(n,v,cost,dist);
printf("\nShortest path from \n");
for(i=0;i<n;i++)
if(i!=v)
printf("\n%d -> %d = %d", v, i, dist[i]);
getch();
}
```

Output:

Enter the number of vertices: 5
Enter the cost matrix <for infinity, enter 999>:
0 5 2 3 999
5 0 4 999 3
2 4 0 999 4
3 999 999 0 999
999 3 4 999 0



Experiment No. 4

CRC CCITT

Aim: For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases a) Without error, b) With error

```
#include<stdio.h>
#include<string.h>
#define N strlen(g)

char t[128], cs[128], g[]="10001000000100001";
int a, e, c, b;

void xor()
{
	for(c=1; c<N; c++)
		cs[c] = ((cs[c] == g[c])? '0' : '1');
}

void crc()
{
	for(e=0; e<N; e++)
		cs[e] = t[e];
	do
	{
		if(cs[0] == '1')
				xor();

		for(c=0; c<N-1; c++)
		{
				cs[c] = cs[c+1];
		}
		cs[c] = t[e++];
	}while(e<=a+N-1);
}

void main()
{
	printf("\nEnter  data t: ");
	scanf("%s", t);

	printf("\nPredefined Generator Polynomial is : %s", g);

	a = strlen(t);
	for(e=a; e<a+N-1; e++)
```

```
            t[e] = '0';
        printf("\nModified t[u] is : %s", t);

        crc();

        printf("\nChecksum is : %s", cs);

        for(e=a; e<a+N-1; e++)
            t[e] = cs[e-a];
        printf("\nFinal Codeword is : %s", t);
        printf("\n\nTest Error detection 1(yes) 0(no) ? : ");
        scanf("%d", &b);
        if(b==1)
        {
            printf("\nEnter position where error is to inserted : ");
            scanf("%d", &e);

            t[e] = (t[e]=='0')?'1':'0';
            printf("Errorneous data : %s\n", t);
        }
        crc();
        for(e=0; (e<N-1)&&(cs[e]!='1');  e++);

        if(e<N-1)
            printf("Error detected.");
        else
            printf("No Error Detected.");
    }
```

Output:

1)

```
Enter data t: 10011

Predefined Generator Polynomial is : 10001000000100001
Modified t[u] is : 10011000000000000000
Checksum is : 0010001001010010
Final Codeword is : 10011001000100101001O

Test Error detection 1(yes) 0(no) ? : 1

Enter position where error is to inserted : 1
Errorneous data : 11011001000100101001O
Error detected.
```

2)

```
Enter data t: 10011

Predefined Generator Polynomial is : 10001000000100001
Modified t[u] is : 10011000000000000000
Checksum is : 00100001001010010
Final Codeword is : 10011001000010010100010

Test Error detection 1(yes) 0(no) ? : 0
No Error Detected.
```

)

Experiment No. 5

Stop & Wait and Sliding Window Protocols

Aim: To implement Stop and Wait protocol and Sliding Window protocol

```
#include<stdio.h>
void sender(void);
void reciever(void);
int frame=0,ack=0,frame_no=1;
int lost_frame=1,lost_ack=2;
int seq_no=0,ack_no=0,timer=0,ready=1,max_frames=3;
void main()
{
// printf("Enter no.of frames\n");
// scanf("%d",&max_frames);
while(frame_no <= max_frames)
{
sender();
sleep(2);
if(frame_no==lost_frame)
{
frame=0;
lost_frame=0;
printf("\nFrame%d has been lost\n",frame_no);
}
reciever();
sleep(2);
if(frame_no==lost_ack && ack==1)
{
ack=0;
lost_ack=0;
printf("\nAck for frame%d has been lost\n",frame_no);
}
if(ack==0)
{
sleep(4);
printf("\nTimer has been expired\n");
timer=1; //time is expired
}
}
}
void sender(void)
{
if(ready)
{
```

```
printf("\nFrame%d is sent with seq no %d\n",frame_no,seq_no);
frame=1;
ready=0;
timer=0; // indicates timer is started
}
else
{
if(timer==1) // indicates timer is expired
{
printf("\nFrame%d is resent with seq no %d\n",frame_no,seq_no);
frame=1;
timer=0; // indicates timer is started
}
if(ack==1)
{
printf("\nAck. for frame%d is recieved\n",frame_no);
seq_no=!seq_no;
ack=0;
frame_no=frame_no+1;
if(frame_no<=max_frames)
{
printf("\nFrame%d is sent with seq no %d\n",frame_no,seq_no);
 frame=1;
 ready=0;
 timer=0; // indicates timer is started
}
}
}
}
void reciever(void)
{
if(frame==1)
{
frame=0; // frame has reached reciever
if(ack_no==seq_no)
{
ack_no=!ack_no;
printf("\nFrame%d is recieved and ack with ack.no. %d sent\n",frame_no,ack_no);
ack=1;
}
else
{
printf("\nFrame%d is duplicate and discarded. Ack with ack.no. %d has been
sent\n",frame_no,ack_no);
ack=1;
}
}
}
```

Output

```
Frame1 is sent with seq no 0
Frame1 has been lost
Timer has been expired
Frame1 is resent with seq no 0
Frame1 is recieved and ack with ack.no. 1 sent
Ack. for frame1 is recieved
Frame2 is sent with seq no 1
Frame2 is recieved and ack with ack.no. 0 sent
Ack for frame2 has been lost
Timer has been expired
Frame2 is resent with seq no 1
Frame2 is duplicate and discarded. Ack with ack.no. 0 has been sent
Ack. for frame2 is recieved
Frame3 is sent with seq no 0
Frame3 is recieved and ack with ack.no. 1 sent
Ack. for frame3 is recieved
Timer has been expired
```

Experiment No. 6

Leaky Bucket Algorithm

Aim: To write a program for congestion control using Leaky bucket algorithm

```
#include<stdio.h>
#include<strings.h>
#include<stdio.h>
int min(int x,int y)
{
if(x<y)
return x;
else
return y;
}
int main()
{
int drop=0,mini,nsec,cap,count=0,i,inp[25],process;
system("clear");
printf("Enter The Bucket Size\n");
scanf("%d",&cap);
printf("Enter The Operation Rate\n");
scanf("%d",&process);
printf("Enter The No. Of Seconds You Want To Stimulate\n");
scanf("%d",&nsec);
for(i=0;i<nsec;i++)
{
printf("Enter The Size Of The Packet Entering At %d sec\n",i+1);
scanf("%d",&inp[i]);
}
printf("\nSecond|Packet Recieved|Packet Sent|Packet Left|Packet Dropped|\n");
printf("-------------------------------------------------------------\n");
for(i=0;i<nsec;i++)
{
count+=inp[i];
if(count>cap)
{
drop=count-cap;
count=cap;
}
printf("%d",i+1);
printf("\t%d",inp[i]);
mini=min(count,process);
printf("\t\t%d",mini);
count=count-mini;
```

```
printf("\t\t%d",count);
printf("\t\t%d\n",drop);
drop=0;
}
for(;count!=0;i++)
{
if(count>cap)
{
drop=count-cap;
count=cap;
}
printf("%d",i+1);
printf("\t0");
mini=min(count,process);
printf("\t\t%d",mini);
count=count-mini;
printf("\t\t%d",count);
printf("\t\t%d\n",drop);
}
}
```

Result:

```
Enter The Bucket Size
5
Enter The Operation Rate
2
Enter The No. Of Seconds You Want To Stimulate
3
Enter The Size Of The Packet Entering At 1 sec
5
Enter The Size Of The Packet Entering At 2 sec
4
Enter The Size Of The Packet Entering At 3 sec
3

Second|Packet Recieved|Packet Sent|Packet Left|Packet Dropped|
-----------------------------------------------------------------
1       5               2             3             0
2       4               2             3             2
3       3               2             3             1
4       0               2             1             0
5       0               1             0             0
```