# Exp#7 NS2 SIMULATION

Date:

A simulator is a device,software or system which behaves or operates like a given system when provided with a set of controlled inputs. The need for simulators is:

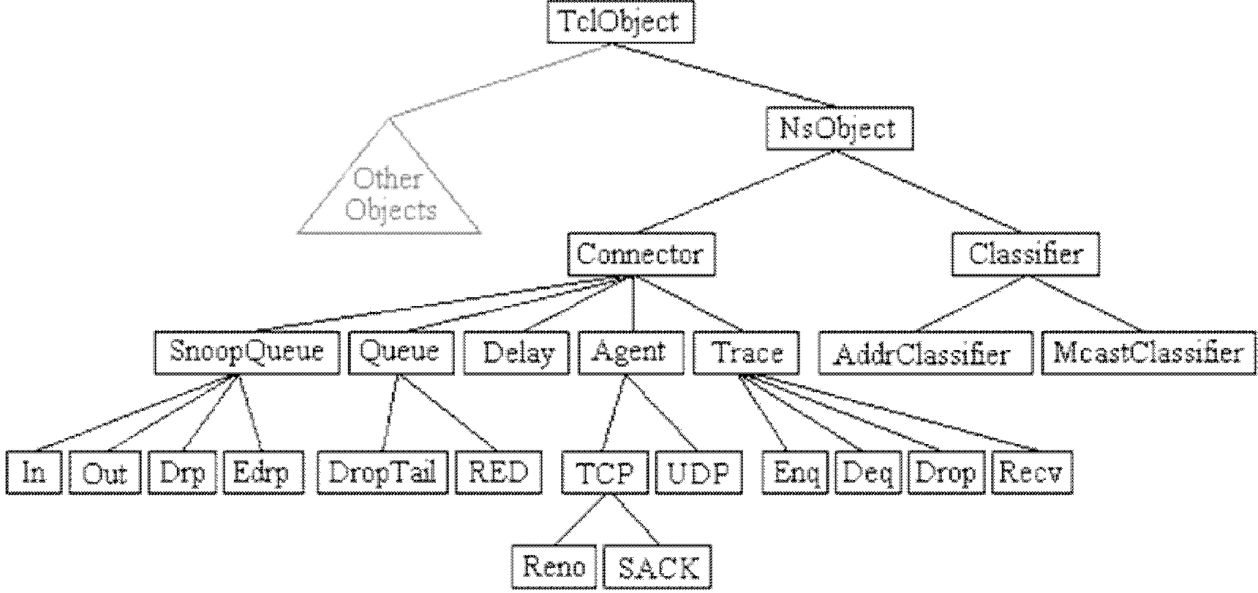
* Provide users with practical feedback such as accuracy,efficiency,cost,etc.,when designing real world systems.
* Permit system designers to study at several different levels of abstraction
* Simulation can give results that are not experimentally measurable with our current level of technology.
* Simulations take the building/rebuilding phase out of the loop by using the model already created in the design phase.
* Effective means for teaching or demonstrating concepts to students.
* A few popular network simulators areNS-2,OPNET,GLOMOSIM,etc.

Network SimulatorNS2

NS2 is anobject-oriented,discrete event driven network simulator developed at UCBerkley written in C++ and OTcl (Object-oriented Tool Command Language). NS is useful for simulating local and wide area networks. NS2 is an open-source simulation tool that primarily runs on Linux (cygwin for Windows). The features of NS2 are:

* Is a discrete event simulator for networking research
* Works at packet level.
* Provide support to simulate bunch of protocols likeTCP,UDP,FTP,etc.
* Simulate wired and wireless network.
* Is a standard experiment environment in research community.

Class Hierarchy



Network Animator(NAM)

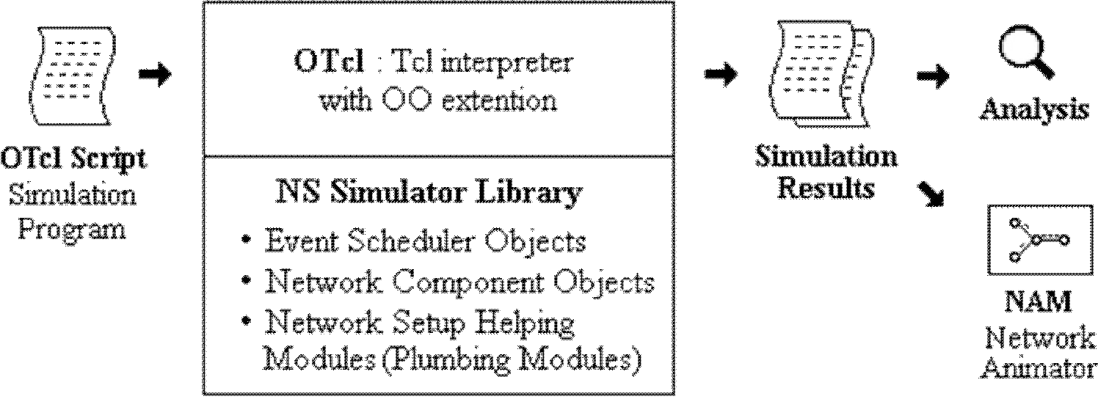
NS together with NAM forms a very powerful set of tools for teaching networking concepts.

With NAM protocols can be visualized as animations. The NAM graphical editor is the latest addition to NAM.With this *editor*,one can create their network topology and simulate various protocols and traffic sources by dragging the mouse.

|  |  |
| --- | --- |
| Create | Visualize |
| * Terrestrial ,satellite eand wireless network with various routing algorithm (DV, LS, PIM, DSR). * Traffic sources like web,ftp,telnet,cbr, and stochastic traffic. * Failures, including deterministic, probabilistic loss, link failure, etc. * Various queuing disciplines(drop-tail, RED, FQ, SFQ, etc.) and QoS | * Packet flow, queue build-up and packet drops. * Protocol behavior: TCP slow start, self- clocking, congestion control, fast retransmit and recovery. * Node movement in wirelesss networks. * Annotations to highlight important events. * Protocol state(e.g.,TCPcwnd). |

NS2 Execution

The overall simulation procedure in NS is shown below.NS is composed of OTcl Script and Interpreter. NS simulation results can be observed through graphs by analyzing the trace file or viewing animations with NAM.



$ns *filename*.tcl

NS2 Program Elements

*Event Scheduler*

1. Creating event scheduler

Set ns[new Simulator]

1. Schedule events

$ns at *time* "*event*"

1. Start scheduler

$ns run

*Creating Network*

1. Create set of Nodes set n0 [$ns node] set n1 [$ns node]
2. Create links and queuing

$ns duplex-link $n0 $n1 *bandwidth delay queue\_type Bandwidth is generally in MB Delay is generally in ms*

*Queue type is either* Drop Tail*,*RED*,*CBQ*,*FQ*,*SFQ*,etc*

$nsduplex-link$n0$n21Mb10msDropTail

1. Layout

$nsduplex-link-op$n0$n2orient*position*

*where position is either* right, right-up, right-down, left, left- up, left-down, up, down

1. Marking flows

$ns color1Blue

$ns color2Red

$udp0 setclass\_1

$udp1 setclass\_2

*Tracing*

1. NAM Trace all links(must succeed scheduler creation)

Setnf [open out.namw]

$ns nam trace-all $nf

1. Trace all links(must succeed scheduler creation)

Set tf[open out.trw]

$ns trace-all $tf

Trace file ouput format

*event,time,from\_node,to\_node,pkttype,pktsize,flags,fid,src\_addr,dst\_addr, seq\_num, pkt\_id*

*where events are* r*received,*+*enqueued,*-*dequeued,*d*dropped*

1. Tracing specific links

$nstrace-queue$n0$n1

$nsnamtrace-queue$n0$n1

*Connection*

1. UDP

set udp [new Agent/UDP] set null [new Agent/Null]

$ns attach-agent$n0$udp0

$ns attach-agent$n1$null

$ns connect$udp0$null

1. TCP

settcp0[new Agent/TCP/Full Tcp]

$tcp0 setwindow\_30

$tcp0 setsegsize\_536

$nsattach-agent$n0$tcp0

setsink0[new Agent/TCP/Full Tcp]

$ns attach-agent$n5$sink0

$sink0 listen

$ns connect$tcp0$sink0

*Traffic Generation*

1. UDP

Set src[new Application/Traffic/*type*]

$src attach-agent$udp0

*Where type is either* CBR,Exponential,Pareto

1. TCP

Set ftp[new Application/FTP]

$ftp attach-agent $tcp

Set telnet[new Application/Telnet]

$telnet attach-agent $tcp

*Finish procedure*

a Flush NS tracing,Close tracing files and execute any post-analysis programs (display results, run NAM, etc)

proc finish {} { global ns nf

$ns

flush-trace close $nf

exec nam out.nam & exit 0

}

Result

Thus simulator NS2 and its basic commands was studied.

# Exp#8a Implementation of UDP performance using simulation tool

Date:

Aim

To study the performance of UDP by simulating as implementation of

network

Algorithm

1. Create a simulator object
2. Define different color for data flows
3. Trace all events in a nam file.
4. Create four nodes *n0*,*n1*,*n2* and *n3*
5. Describe their layout topology
6. Specify the link capacity between nodes
7. Monitor queue on the link *n2* to *n3* vertically 90°
8. Create a UDP agent s*udp0*,*udp1*and attach it to nodes *n0* and *n1* respectively
9. Create a CBR traffic *cbr0*,*cbr1* and attach it to *udp0* and *udp1* respectively
10. Create a traffic sink and attach it to node *n3*
11. Connect sources to the sink
12. Label the nodes
13. Schedule *cbr0* to start at0.5 and stop at4.5seconds
14. Schedule *cbr1*to start at1.0 and stop at4.0seconds
15. Call finish procedure at 5.0seconds
16. Run the simulation
17. Execute NAM on the trace file
18. Observe simulated events on theNAM and packet flow on link n2 to n3
19. Stop

Program

# Study of UDP performance - UDP.tcl

# Create a simulator object

set ns [new Simulator]

# Define different colors for data flows in NAM

$ns color 1 Blue

$ns color 2 Red

# Open the NAM trace file

set nf [open out.nam w]

$ns namtrace-all $nf

# Create four nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

# Create links between the nodes

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

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

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

# Specify layout orientation (for NAM visualization)

$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

# Monitor the queue for the link n2-n3

$ns duplex-link-op $n2 $n3 queuePos 0.5

# Create a UDP agent and attach it to node n0

set udp0 [new Agent/UDP]

$udp0 set class\_ 1

$ns attach-agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.005

$cbr0 attach-agent $udp0

# Create a UDP agent and attach it to node n1

set udp1 [new Agent/UDP]

$udp1 set class\_ 2

$ns attach-agent $n1 $udp1

# Create a CBR traffic source and attach it to udp1

set cbr1 [new Application/Traffic/CBR]

$cbr1 set packetSize\_ 500

$cbr1 set interval\_ 0.005

$cbr1 attach-agent $udp1

# Create a Null agent (traffic sink) and attach it to node n3

set null0 [new Agent/Null]

$ns attach-agent $n3 $null0

# Connect UDP agents to the traffic sink

$ns connect $udp0 $null0

$ns connect $udp1 $null0

# Label the nodes for NAM

$ns at 0.0 "$n0 label Sender1"

$ns at 0.0 "$n1 label Sender2"

$ns at 0.0 "$n2 label Router"

$ns at 0.0 "$n3 label Receiver"

# Schedule traffic start/stop times

$ns at 0.5 "$cbr0 start"

$ns at 1.0 "$cbr1 start"

$ns at 4.0 "$cbr1 stop"

$ns at 4.5 "$cbr0 stop"

# Define the finish procedure

proc finish {} {

global ns nf

$ns flush-trace

close $nf

exec nam out.nam &

exit 0

}

# Call the finish procedure after simulation time

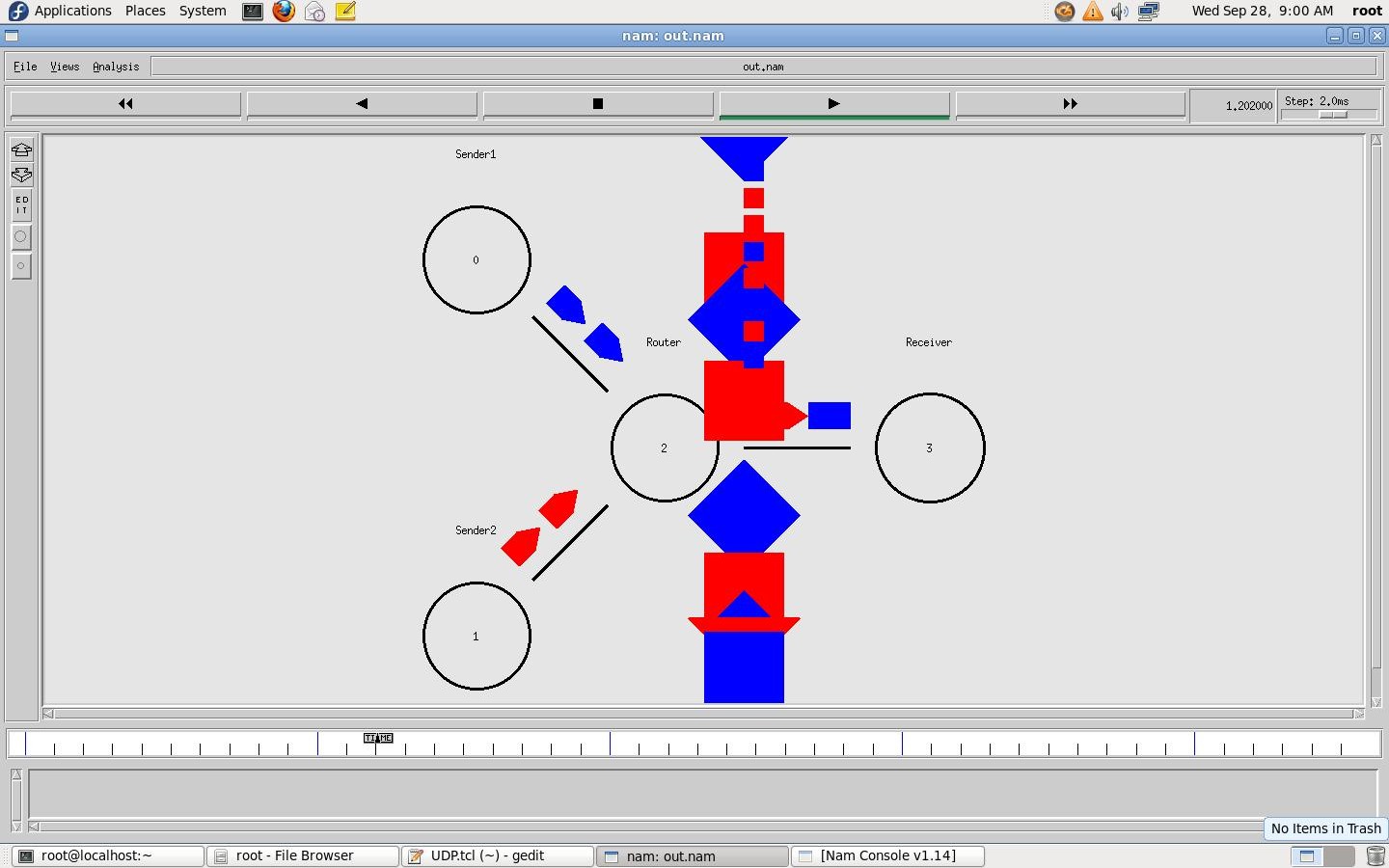
$ns at 5.0 "finish"

# Run the simulation

$ns run

Output

$ ns UDP.tcl



Result:

Thus the performance of UDP and basic network terminologies were studied using

NS2

# Exp#8b Implementation of TCP performance using simulation tool

Date:

Aim

To study the performance of a TCP network with drop tail queue mechanismon the

gateway

Algorithm

1. Create a simulator object
2. Define different flows for dataflows
3. Trace all events in a nam file and text file
4. Create source nodes(*s1*,*s2*,*s3*),gateway(*G*)and receiver(*r*)
5. Describe their layout topology
6. Specify the link between nodes
7. Definethequeuesizebetweennodes*G*and*r*as5
8. Monitor queue on all links vertically 90°
9. Create TCP agents*tcp1*,*tcp2*,*tcp3*and attach it to nodes*s1*,*s2*and*s3* respectively
10. Create three TCP sinks and attach it to node *r*
11. Connect traffic sources to the sink
12. Create FTP agents [*ftp1*,*ftp2*,*ftp3*](ftp://ftp1,ftp2,ftp3)and attach it to *tcp1*,*tcp2*and *tcp3* respectively
13. Label the nodes at start time
14. Schedule *ftp1*,*ftp2*,*ftp3* to start at 0.1and stop at 5.0seconds
15. Call *finish* procedure at 5.25 seconds
16. Run the simulation
17. Execute NAM on the trace file
18. Observe the simulated events on the NAM editor and packet flow on link G to r
19. View the trace file and analyse the events
20. Stop

Program

# Study of TCP performance - TCP.tcl

# Create a simulator object

set ns [new Simulator]

# Open trace file for analysis

set f [open droptail-queue-out.tr w]

$ns trace-all $f

# Open the NAM trace file

set nf [open droptail-queue-out.nam w]

$ns namtrace-all $nf

# Create nodes: 3 senders, 1 gateway, 1 receiver

set s1 [$ns node]

set s2 [$ns node]

set s3 [$ns node]

set G [$ns node]

set r [$ns node]

# Define different colors for data flows

$ns color 1 red

$ns color 2 SeaGreen

$ns color 3 blue

# Create duplex links with DropTail queues

$ns duplex-link $s1 $G 6Mb 10ms DropTail

$ns duplex-link $s2 $G 6Mb 10ms DropTail

$ns duplex-link $s3 $G 6Mb 10ms DropTail

$ns duplex-link $G $r 3Mb 10ms DropTail

# Define layout in NAM

$ns duplex-link-op $s1 $G orient right-up

$ns duplex-link-op $s2 $G orient right

$ns duplex-link-op $s3 $G orient right-down

$ns duplex-link-op $G $r orient right

# Set queue size for the G → r link

$ns queue-limit $G $r 5

# Enable queue monitoring for all links

$ns duplex-link-op $s1 $G queuePos 0.5

$ns duplex-link-op $s2 $G queuePos 0.5

$ns duplex-link-op $s3 $G queuePos 0.5

$ns duplex-link-op $G $r queuePos 0.5

# Create TCP agents and attach them to senders

set tcp1 [new Agent/TCP/Reno]

$ns attach-agent $s1 $tcp1

$tcp1 set window\_ 8

$tcp1 set fid\_ 1

set tcp2 [new Agent/TCP/Reno]

$ns attach-agent $s2 $tcp2

$tcp2 set window\_ 8

$tcp2 set fid\_ 2

set tcp3 [new Agent/TCP/Reno]

$ns attach-agent $s3 $tcp3

$tcp3 set window\_ 4

$tcp3 set fid\_ 3

# Create TCP sink agents at receiver

set sink1 [new Agent/TCPSink]

set sink2 [new Agent/TCPSink]

set sink3 [new Agent/TCPSink]

$ns attach-agent $r $sink1

$ns attach-agent $r $sink2

$ns attach-agent $r $sink3

# Connect senders to sinks

$ns connect $tcp1 $sink1

$ns connect $tcp2 $sink2

$ns connect $tcp3 $sink3

# Create FTP applications over TCP agents

set ftp1 [new Application/FTP]

$ftp1 attach-agent $tcp1

set ftp2 [new Application/FTP]

$ftp2 attach-agent $tcp2

set ftp3 [new Application/FTP]

$ftp3 attach-agent $tcp3

# Label nodes in NAM

$ns at 0.0 "$s1 label Sender1"

$ns at 0.0 "$s2 label Sender2"

$ns at 0.0 "$s3 label Sender3"

$ns at 0.0 "$G label Gateway"

$ns at 0.0 "$r label Receiver"

# Schedule FTP traffic start and stop

$ns at 0.1 "$ftp1 start"

$ns at 0.1 "$ftp2 start"

$ns at 0.1 "$ftp3 start"

$ns at 5.0 "$ftp1 stop"

$ns at 5.0 "$ftp2 stop"

$ns at 5.0 "$ftp3 stop"

# Define finish procedure

proc finish {} {

global ns f nf

$ns flush-trace

close $f

close $nf

puts "Simulation completed. Running NAM..."

exec nam droptail-queue-out.nam &

exit 0

}

# Schedule finish after simulation time

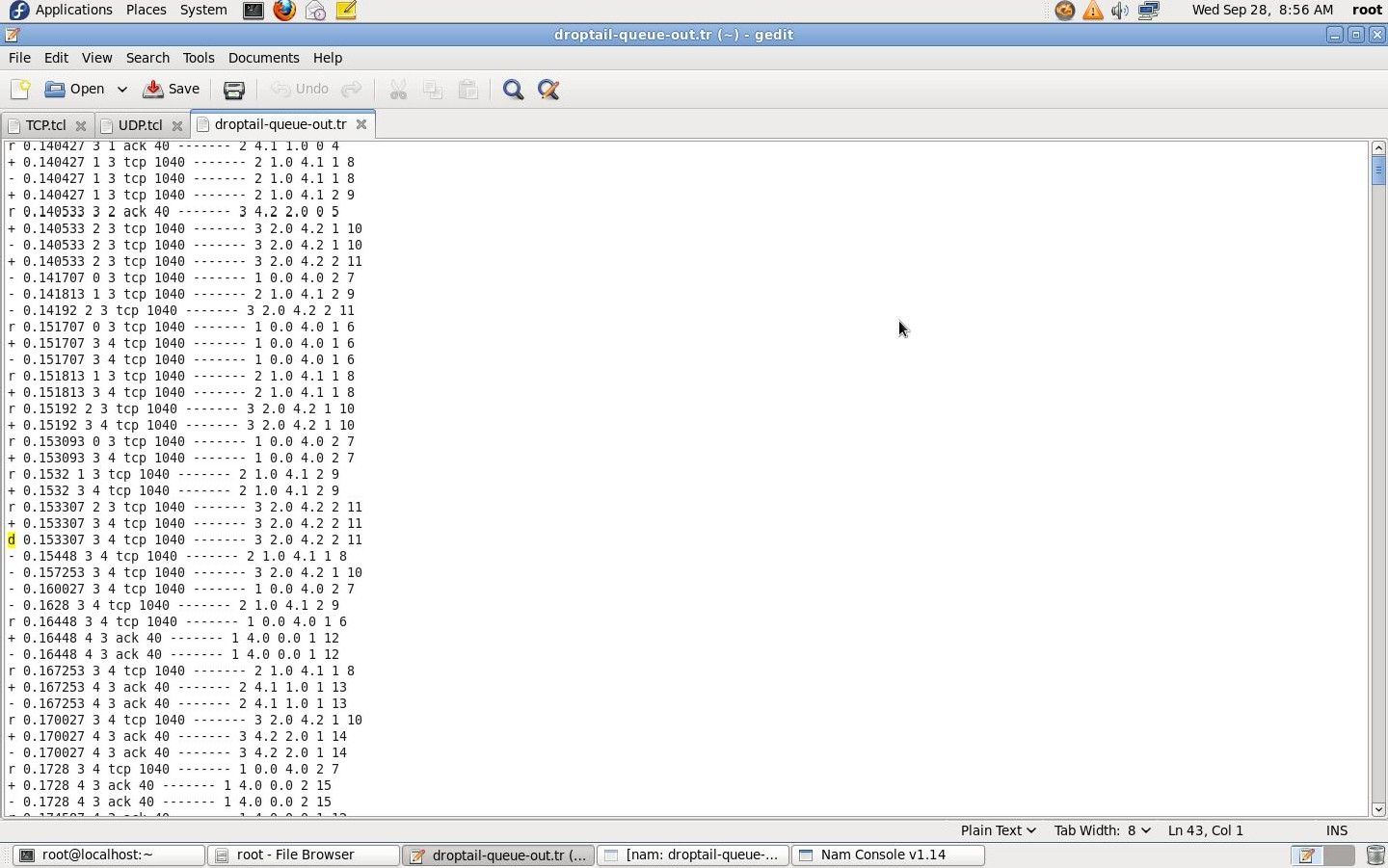
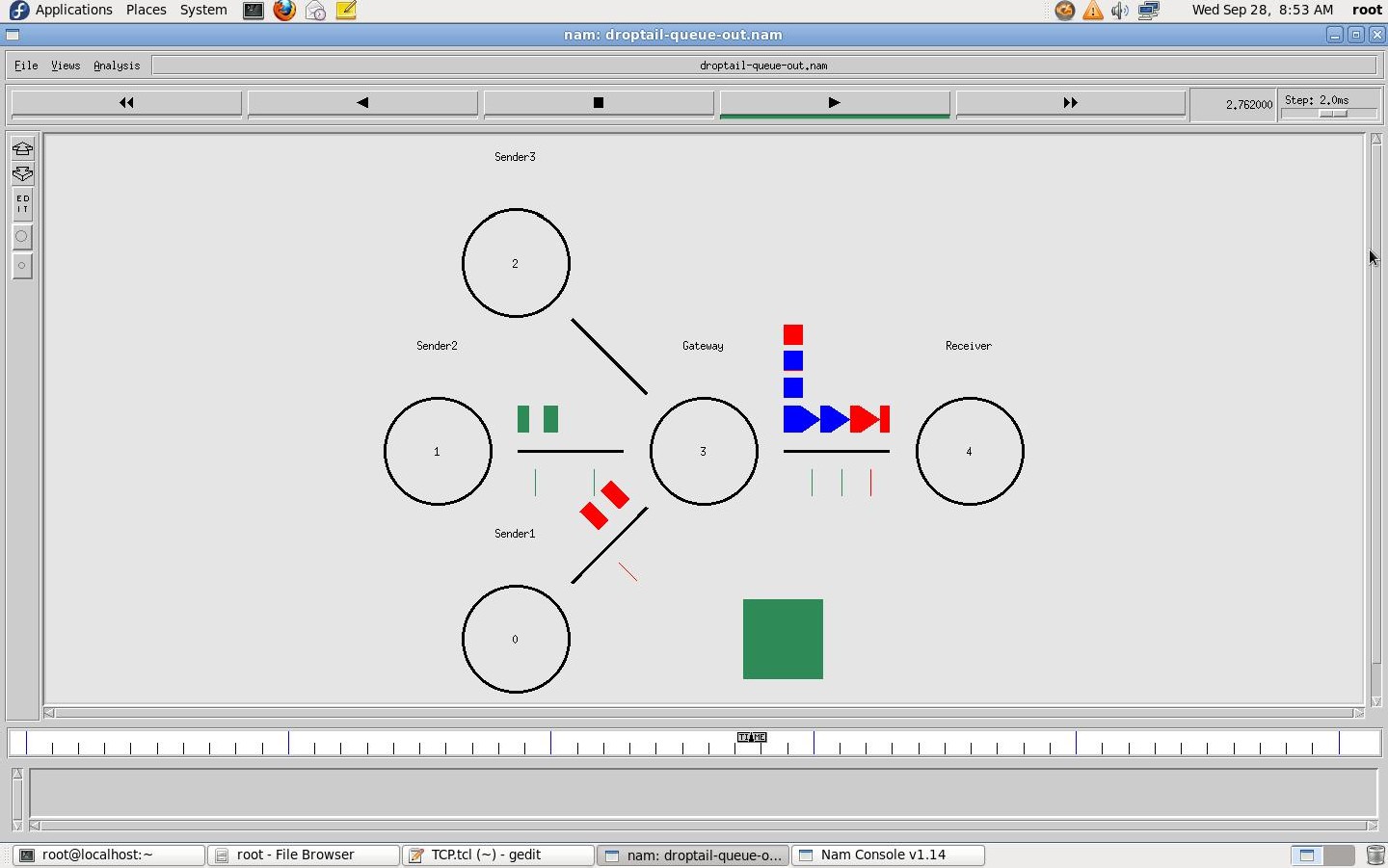
$ns at 5.25 "finish"

# Run the simulation

$ns run

Output

$ ns TCP.tcl



Result

Thus the behaviour of TCP was observed and the basic terminologies of TCP transmission were understood.

Exp#9a Simulation of Distance Vector Routing algorithm.

# 

Date:

Aim

To simulate a link failure and to observe distance vector routing protocol in action.

Algorithm

1. Create a simulator object
2. Set routing protocol l to Distance Vector routing
3. Trace packets on all links onto NAM trace and text trace file
4. Define finish procedure to close files,flush tracing and run NAM
5. Create eight nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology as aoctagon
8. Add UDP agent for node n1
9. Create CBR traffic on top of UDP and set traffic parameters.
10. Add a sink agent to node n4
11. Connect source and the sink
12. Schedule events as follows:
    1. Start traffic flow at 0.5
    2. Down the linkn3-n4at1.0
    3. Up the linkn3-n4at2.0
    4. Stop traffic at 3.0
    5. Call finish procedure at 5.0
13. Start the scheduler
14. Observe the traffic route when link is up and down
15. View the simulated events and trace file analyze it
16. Stop

Program

# Distance Vector Routing Protocol - distvect.tcl

# Create a simulator object

set ns [new Simulator]

# Use distance vector routing

$ns rtproto DV

# Open the NAM trace file

set nf [open out.nam w]

$ns namtrace-all $nf

# Open the general trace file

set nt [open trace.tr w]

$ns trace-all $nt

# Define 'finish' procedure

proc finish {} {

global ns nf nt

$ns flush-trace

close $nf

close $nt

puts "Simulation completed. Running NAM..."

exec nam -a out.nam &

exit 0

}

# Create 8 nodes

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]

# Create duplex links (1Mb, 10ms)

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

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

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

$ns duplex-link $n4 $n5 1Mb 10ms DropTail

$ns duplex-link $n5 $n6 1Mb 10ms DropTail

$ns duplex-link $n6 $n7 1Mb 10ms DropTail

$ns duplex-link $n7 $n8 1Mb 10ms DropTail

$ns duplex-link $n8 $n1 1Mb 10ms DropTail

# Octagon layout in NAM

$ns duplex-link-op $n1 $n2 orient left-up

$ns duplex-link-op $n2 $n3 orient up

$ns duplex-link-op $n3 $n4 orient right-up

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

$ns duplex-link-op $n5 $n6 orient right-down

$ns duplex-link-op $n6 $n7 orient down

$ns duplex-link-op $n7 $n8 orient left-down

$ns duplex-link-op $n8 $n1 orient left

# Create a UDP agent at node n1 (source)

set udp0 [new Agent/UDP]

$ns attach-agent $n1 $udp0

# Create a CBR traffic source and attach it to UDP agent

set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize\_ 500

$cbr0 set interval\_ 0.005

$cbr0 attach-agent $udp0

# Create a Null agent (traffic sink) at node n4 (destination)

set null0 [new Agent/Null]

$ns attach-agent $n4 $null0

# Connect UDP source to sink

$ns connect $udp0 $null0

# Add labels in NAM

$ns at 0.0 "$n1 label Source"

$ns at 0.0 "$n4 label Destination"

# Start and stop CBR traffic

$ns at 0.5 "$cbr0 start"

$ns at 4.5 "$cbr0 stop"

# Simulate link failure and recovery

$ns rtmodel-at 1.0 down $n3 $n4

$ns rtmodel-at 2.0 up $n3 $n4

# Call the finish procedure

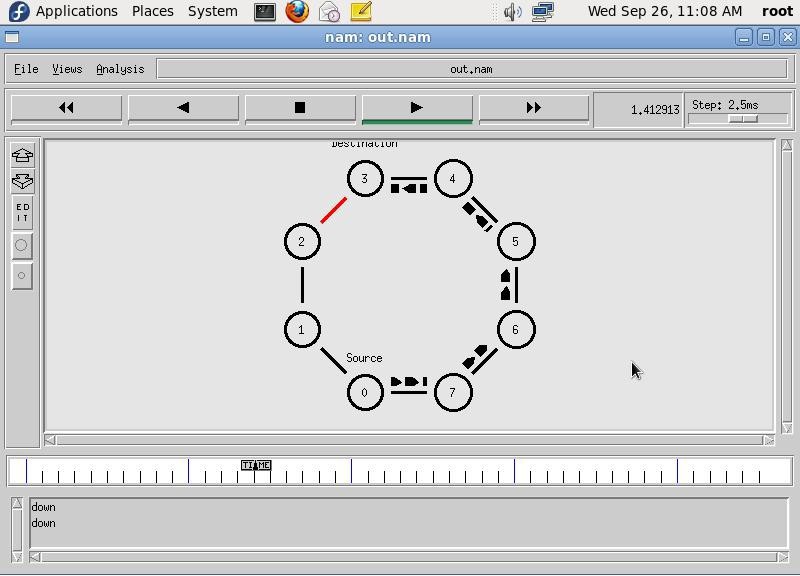
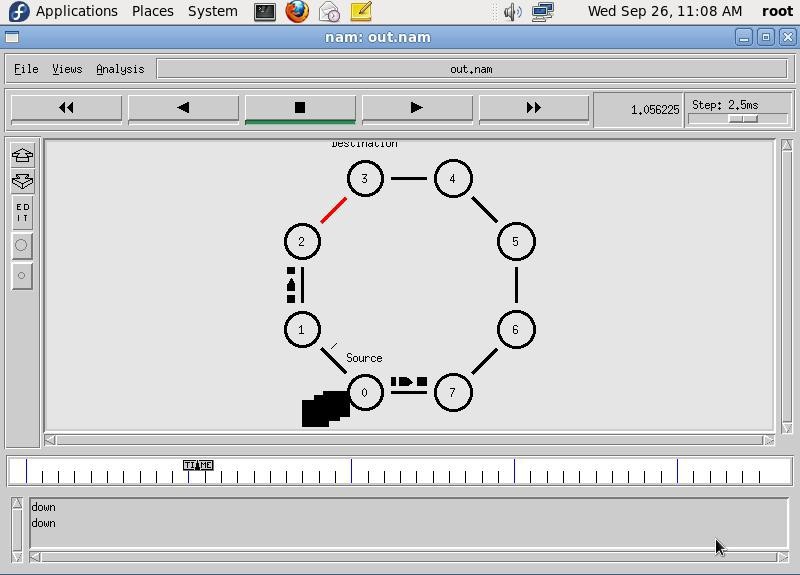
$ns at 5.0 "finish"

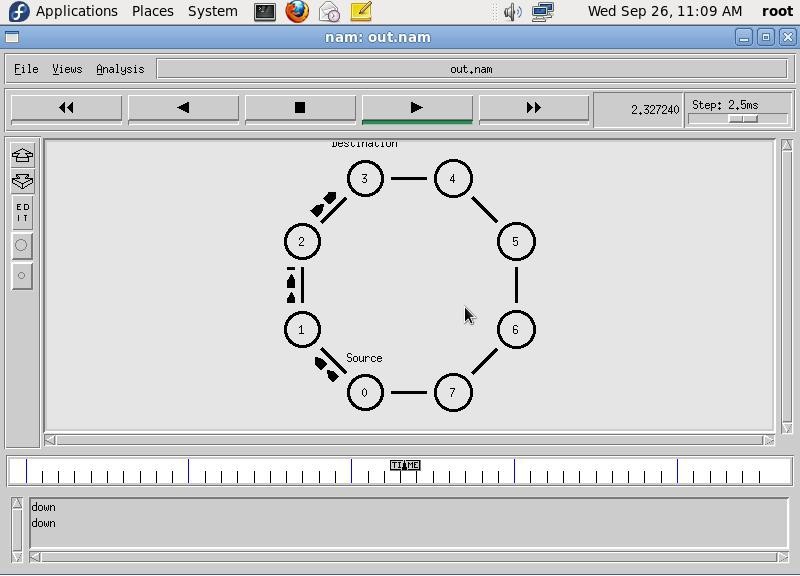
# Run the simulation

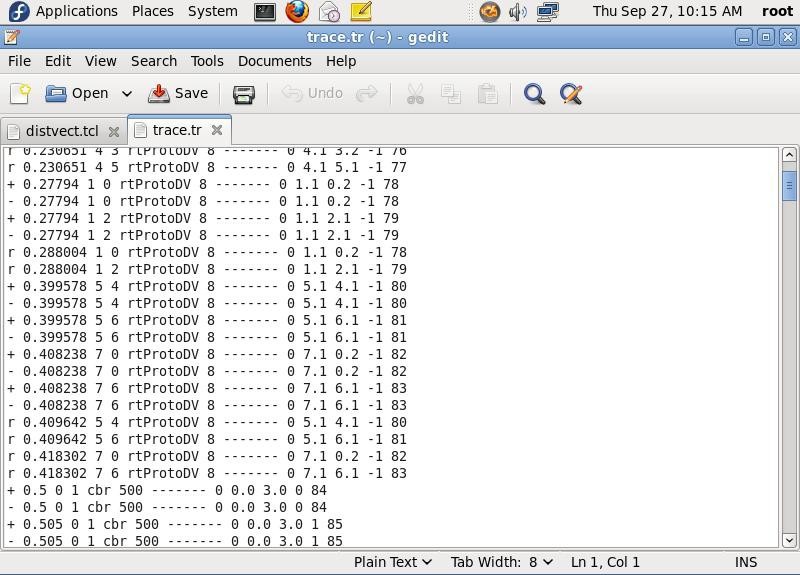
$ns run

OUTPUT:

$ ns distvect.tcl







Result

Thus, performance of distance vector protocol and routing path was studied using NS2.

Exp.No.9b Simulation of Link State Routing algorithm.

Date:

Aim

To simulate a link failure and to observe link state routing protocol inaction.

Algorithm

1. Create a simulator object
2. Set routing protocol to Link State routing
3. Trace packets on all links on to NAM trace and text trace file
4. Define finish procedure to close files,flush tracing and run NAM
5. Create twelve nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology in an adhoc manner.
8. Create CBR traffic on top of UDP and set traffic parameters.
9. Create source and sink and connect them
10. Schedule events as follows:
    1. Start traffic flows at1.0and2.0
    2. Down the link n5-n11at10.0and restore it at 30.0
    3. Down the lin kn7-n6 at15.0and restore it at20.0
    4. Call finish procedure at 45.0
11. Start the scheduler
12. Observe the traffic route when link is upand down
13. View the simulated events and trace file analyze it
14. Stop

Program

# Throughput simulation with link failures - throughput.tcl

# Create simulator object

set ns [new Simulator]

# Open trace files

set nr [open thro.tr w]

$ns trace-all $nr

set nf [open thro.nam w]

$ns namtrace-all $nf

# Finish procedure

proc finish {} {

global ns nr nf

$ns flush-trace

close $nr

close $nf

exec nam thro.nam &

exit 0

}

# Create 12 nodes

for {set i 0} {$i < 12} {incr i} {

set n($i) [$ns node]

}

# Linear links from node 0 to node 8

for {set i 0} {$i < 8} {incr i} {

$ns duplex-link $n($i) $n([expr $i + 1]) 1Mb 10ms DropTail

}

# Additional links forming alternate paths

$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail

$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail

$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail

$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail

$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail

$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail

# Create first UDP flow (n0 → n5)

set udp0 [new Agent/UDP]

$ns attach-agent $n(0) $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 $n(5) $null0

$ns connect $udp0 $null0

# Create second UDP flow (n1 → n5)

set udp1 [new Agent/UDP]

$ns attach-agent $n(1) $udp1

set cbr1 [new Application/Traffic/CBR]

$cbr1 set packetSize\_ 500

$cbr1 set interval\_ 0.005

$cbr1 attach-agent $udp1

set null1 [new Agent/Null]

$ns attach-agent $n(5) $null1

$ns connect $udp1 $null1

# Assign flow IDs and colors for NAM

$udp0 set fid\_ 1

$udp1 set fid\_ 2

$ns color 1 Red

$ns color 2 Green

# Use Link-State routing

$ns rtproto LS

# Simulate dynamic link changes

$ns rtmodel-at 10.0 down $n(11) $n(5)

$ns rtmodel-at 15.0 down $n(7) $n(6)

$ns rtmodel-at 20.0 up $n(7) $n(6)

$ns rtmodel-at 30.0 up $n(11) $n(5)

# Start traffic flows

$ns at 1.0 "$cbr0 start"

$ns at 2.0 "$cbr1 start"

# End simulation

$ns at 45.0 "finish"

# Run the simulation

$ns run

OUTPUT:

$ ns ls.tcl

