# Exp#6a ARP Client/Server

Date:

Aim

To know the physical address of a host when its logical address is known using ARP

protocol.

Algorithm

Target/Server

1. Create a server socket.
2. Accept client connection.
3. Read IPaddress from the client request
4. Check its configuration file and compare with its logical address.
5. If there is a match,send the host physical address.
6. Stop

Client

1. Create a socket.
2. Send IP address to the target machine
3. Receive target's response
4. If It is a MAC address then display it and goto step 6
5. Display"Host not found"
6. Stop

Program

// ARP Server –ArpServer.java

import java.io.\*;

import java.net.\*;

public class ArpServer {

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(2500)) {

System.out.println("ARP Server started on port 2500...");

try (Socket clientSocket = serverSocket.accept()) {

System.out.println("Client connected.");

BufferedReader br = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintStream ps = new PrintStream(clientSocket.getOutputStream());

String requestedIP = br.readLine().trim();

System.out.println("Requested IP: " + requestedIP);

String macAddress = getMacFromIpconfig(requestedIP);

if (macAddress != null) {

ps.println("MAC Address: " + macAddress);

} else {

ps.println("IP address not found or MAC address not available.");

}

}

} catch (IOException e) {

System.err.println("Server Error: " + e);

}

}

// Helper method to run ipconfig and extract MAC address

private static String getMacFromIpconfig(String ipAddress) {

try {

Process process = Runtime.getRuntime().exec("ipconfig /all");

BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));

String line;

boolean foundIPBlock = false;

String macAddress = null;

while ((line = reader.readLine()) != null) {

line = line.trim();

// Look for the requested IP address

if (line.contains(ipAddress)) {

foundIPBlock = true;

}

// If we're in the relevant block and find "Physical Address", capture it

if (foundIPBlock && line.toLowerCase().startsWith("physical address")) {

int index = line.indexOf(":");

if (index != -1) {

macAddress = line.substring(index + 1).trim();

break;

}

}

// Reset if a new interface block starts (indicated by an empty line)

if (line.isEmpty()) {

foundIPBlock = false;

}

}

reader.close();

return macAddress;

} catch (IOException e) {

System.err.println("Failed to execute ipconfig: " + e);

return null;

}

}

}

// ARP Client -- ArpClient.java

import java.io.\*;

import java.net.\*;

public class ArpClient {

public static void main(String[] args) {

try {

// Connect to ARP server on localhost and port 2500

Socket client = new Socket("localhost", 2500);

System.out.println("Connected to ARP Server.");

// Input from user

BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

// Output stream to server

PrintStream out = new PrintStream(client.getOutputStream());

// Input stream from server

BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));

// Prompt user for IP address

System.out.print("Enter the IP address: ");

String ipAddress = userInput.readLine();

// Send IP address to server

out.println(ipAddress);

// Receive MAC address or error from server

String response = in.readLine();

// Display the response

if (response == null || response.trim().isEmpty()) {

System.out.println("Host does not exist or MAC address not found.");

} else {

System.out.println("Response from Server: " + response);

}

// Close all connections

in.close();

out.close();

client.close();

userInput.close();

} catch (IOException e) {

System.err.println("Client Error: " + e.getMessage());

}

}

}

**Output:**

Server

$ javac ArpServer.java

$ java ArpServer Server started

**eth0 Link encap:Ethernet HWaddr B8:AC:6F:1B:AB:06**

**inet addr:172.16.12.251 Bcast:172.255.255.255 Mask:255.0.0.0**

**inet6 addr: fe80::baac:6fff:fe1b:ab06/64 Scope:Link UP BROADCAST RUNNING MULTICAST**

**MTU:1500 Metric:1**

**RX packets:450 errors:0 dropped:0 overruns:0 frame:0 TX packets:127 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:48118 (46.9 KiB) TX bytes:21025 (20.5 KiB)**

**Interrupt:16**

Client

$ javac ArpClient.java

$ java ArpClient

Enter the IP address : 172.16.12.251 Physical Address B8:AC:6F:1B:AB:06

Result

Thus using ARP protocol,server’sMAC address is obtained

# Exp#6b RARP Client/Server

Date:

Aim

To know the logical address of a host when its physical address is known using RARP

protocol.

Algorithm

Target/Server

1. Create a server socket.
2. Accept client connection.
3. Read MAC address from the client request
4. Check its configuration file and compare with its physical address.
5. If there is a match,send the host logical address.
6. Stop

Client

1. Create a socket.
2. Send physical address to the target machine
3. Receive target's response
4. If it is a Ip address then display it and goto step6
5. Display"Host not found"
6. Stop

Program

//RARPServer--RarpServer.java

import java.io.\*;

import java.net.\*;

public class RarpServer {

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(2500)) {

System.out.println("RARP Server started. Waiting for client...");

try (Socket clientSocket = serverSocket.accept()) {

System.out.println("Client connected.");

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintStream out = new PrintStream(clientSocket.getOutputStream());

String macInput = in.readLine().trim().toLowerCase().replace("-", ":"); // Normalize MAC

System.out.println("Received MAC address: " + macInput);

// Run ipconfig /all

Process process = Runtime.getRuntime().exec("ipconfig /all");

BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));

String line;

String currentIP = null;

String foundIP = null;

while ((line = reader.readLine()) != null) {

line = line.trim();

// Extract IP address

if (line.toLowerCase().startsWith("ipv4 address") || line.toLowerCase().contains("ip address")) {

int idx = line.indexOf(":");

if (idx != -1) {

currentIP = line.substring(idx + 1).trim().replace("(Preferred)", "").trim();

}

}

// Extract MAC address

if (line.toLowerCase().startsWith("physical address")) {

int idx = line.indexOf(":");

if (idx != -1) {

String mac = line.substring(idx + 1).trim().toLowerCase().replace("-", ":");

if (mac.equals(macInput)) {

foundIP = currentIP;

break;

}

}

}

}

reader.close();

if (foundIP != null) {

out.println("IP Address: " + foundIP);

} else {

out.println("IP address not found for given MAC.");

}

}

} catch (IOException e) {

System.err.println("RARP Server Error: " + e.getMessage());

}

}

}

//RARPClient--RarpClient.java

import java.io.\*;

import java.net.\*;

public class RarpClient {

public static void main(String[] args) {

try {

// Allow optional server IP as command-line argument

String serverIP = (args.length > 0) ? args[0] : "localhost";

// Connect to the RARP server

Socket client = new Socket(serverIP, 2500);

System.out.println("Connected to RARP Server at " + serverIP);

// Reader for user input (keyboard)

BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

// Output stream to server

PrintStream out = new PrintStream(client.getOutputStream());

// Input stream from server

BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));

// Prompt user to enter MAC address

System.out.print("Enter the MAC address (e.g., 00-1A-2B-3C-4D-5E or 00:1A:2B:3C:4D:5E): ");

String macAddress = userInput.readLine().trim();

// Optional: validate MAC address format

if (!macAddress.matches("(?i)([0-9A-F]{2}[:-]){5}[0-9A-F]{2}")) {

System.out.println("Invalid MAC address format.");

client.close();

return;

}

// Send MAC address to server

out.println(macAddress);

// Read IP address (response) from server

String response = in.readLine();

// Display result

if (response == null || response.isEmpty()) {

System.out.println("No response from server.");

} else if (response.toLowerCase().contains("not found")) {

System.out.println("IP address not found for given MAC.");

} else {

System.out.println("IP Address: " + response);

}

// Close resources

out.close();

in.close();

userInput.close();

client.close();

} catch (IOException e) {

System.err.println("Client error: " + e.getMessage());

}

}

}

Server

$ javac RarpServer.java

$ java RarpServer Server started

**eth0 Link encap:Ethernet HWaddr B8:AC:6F:1B:AB:06**

**inet addr:172.16.12.251 Bcast:172.255.255.255 Mask:255.0.0.0**

**inet6 addr: fe80::baac:6fff:fe1b:ab06/64 Scope:Link UP BROADCAST RUNNING MULTICAST**

**MTU:1500 Metric:1**

**RX packets:450 errors:0 dropped:0 overruns:0 frame:0 TX packets:127 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:48118 (46.9 KiB) TX bytes:21025 (20.5 KiB)**

**Interrupt:16**

Client

$ javac RarpClient.java

$ java RarpClient

Enter the physical address : B8:AC:6F:1B:AB:06 Logical Address 172.16.12.251

Result

Thus using RARP protocol, IP address of the server is obtained.

# 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:

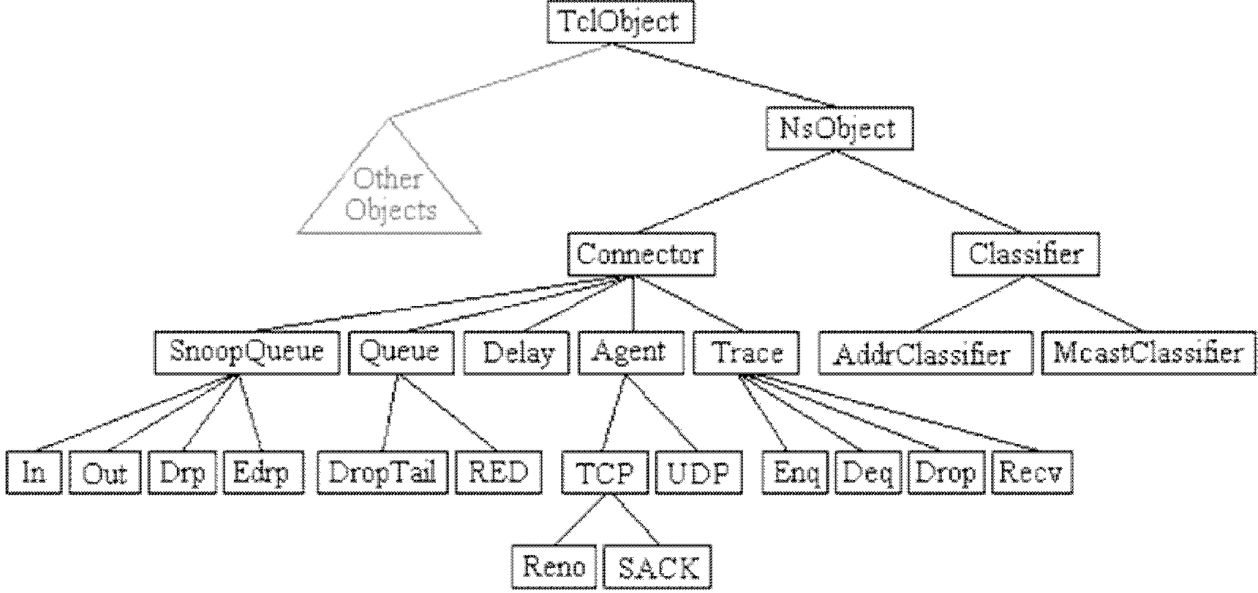
* Provide users with practical feedback such as accuracy,efficiency,cost,etc.,when designing real world systems.
* Permit system designers to study at several different levels of abstraction
* Simulation can give results that are not experimentally measurable with our current level of technology.
* Simulations 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)

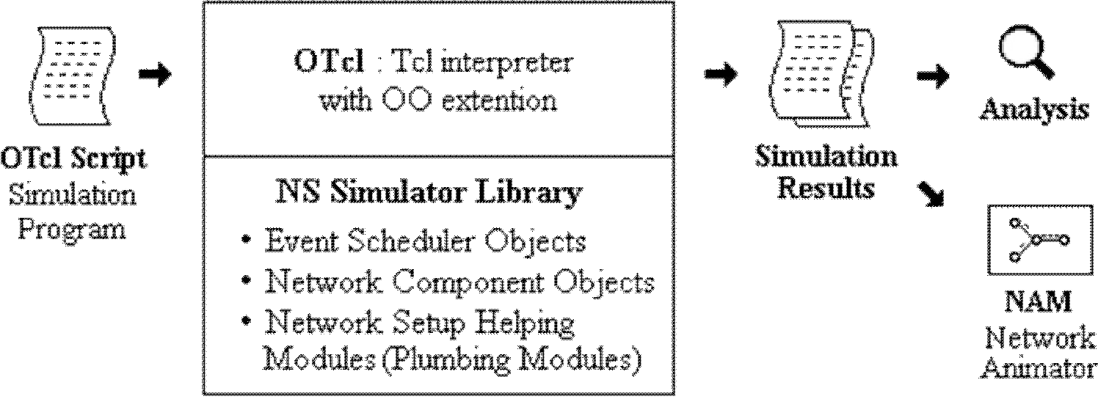
NS together with NAM forms a very powerful set of tools for teaching networking concepts.

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 Study of UDP Performance

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

setns[newSimulator]

#Define different colors for dataflows

$nscolor1Blue

$nscolor2Red

#Open the namtrace file set nf [open out.nam w]

$nsnamtrace-all$nf

#Create four nodes set n0 [$nsnode]set n1 [$ns node] set n2 [$ns node] set n3 [$ns node]

#Create links between the nodes

$nsduplex-link$n0$n21Mb10msDropTail

$nsduplex-link$n1$n21Mb10msDropTail

$nsduplex-link$n3$n21Mb10msSFQ

#Specify layout to fnodes

$nsduplex-link-op$n0$n2orientright-down

$nsduplex-link-op$n1$n2orientright-up

$nsduplex-link-op$n2$n3orientright

#Monitor the queue for the link2—3 vertically

$nsduplex-link-op$n2$n3queuePos0.5

#Createa UDP agent and attach it to node n0 set udp0 [new Agent/UDP]

$udp0setclass\_1

$nsattach-agent$n0$udp0

#Createa CBR traffic source and attach it to udp0 set cbr0 [new Application/Traffic/CBR]

$cbr0setpacketSize\_500

$cbr0setinterval\_0.005

$cbr0attach-agent$udp0

#Createa UDP agent and attach it to node n1 set udp1 [new Agent/UDP]

$udp1setclass\_2

$nsattach-agent$n1$udp1

#Createa CBR traffic source and attach it to udp1 set cbr1 [new Application/Traffic/CBR]

$cbr1setpacketSize\_500

$cbr1setinterval\_0.005

$cbr1attach-agent$udp1

#Createa Null agent(a traffic sink)and attach it to node n3 set null0 [new Agent/Null]

$nsattach-agent$n3$null0

#Connect traffic sources with the traffic sink

$nsconnect$udp0$null0

$nsconnect$udp1$null0

#Define finish procedureprocfinish

**{}{**

globalnsnf

$nsflush-trace

#Closethetracefile close $nf

#Executenamonthetracefile exec nam -a out.nam &

exit0

**}**

#Define label for nodes

$nsat0.0"$n0labelSender1"

$nsat0.0"$n1labelSender2"

$nsat0.0"$n2labelRouter"

$nsat0.0"$n3labelReceiver"

#Schedule events for the CBR agents

$nsat0.5"$cbr0start"

$nsat1.0"$cbr1start"

$nsat4.0"$cbr1stop"

$nsat4.5"$cbr0stop"

#Call finish procedure after 5seconds of simulation time

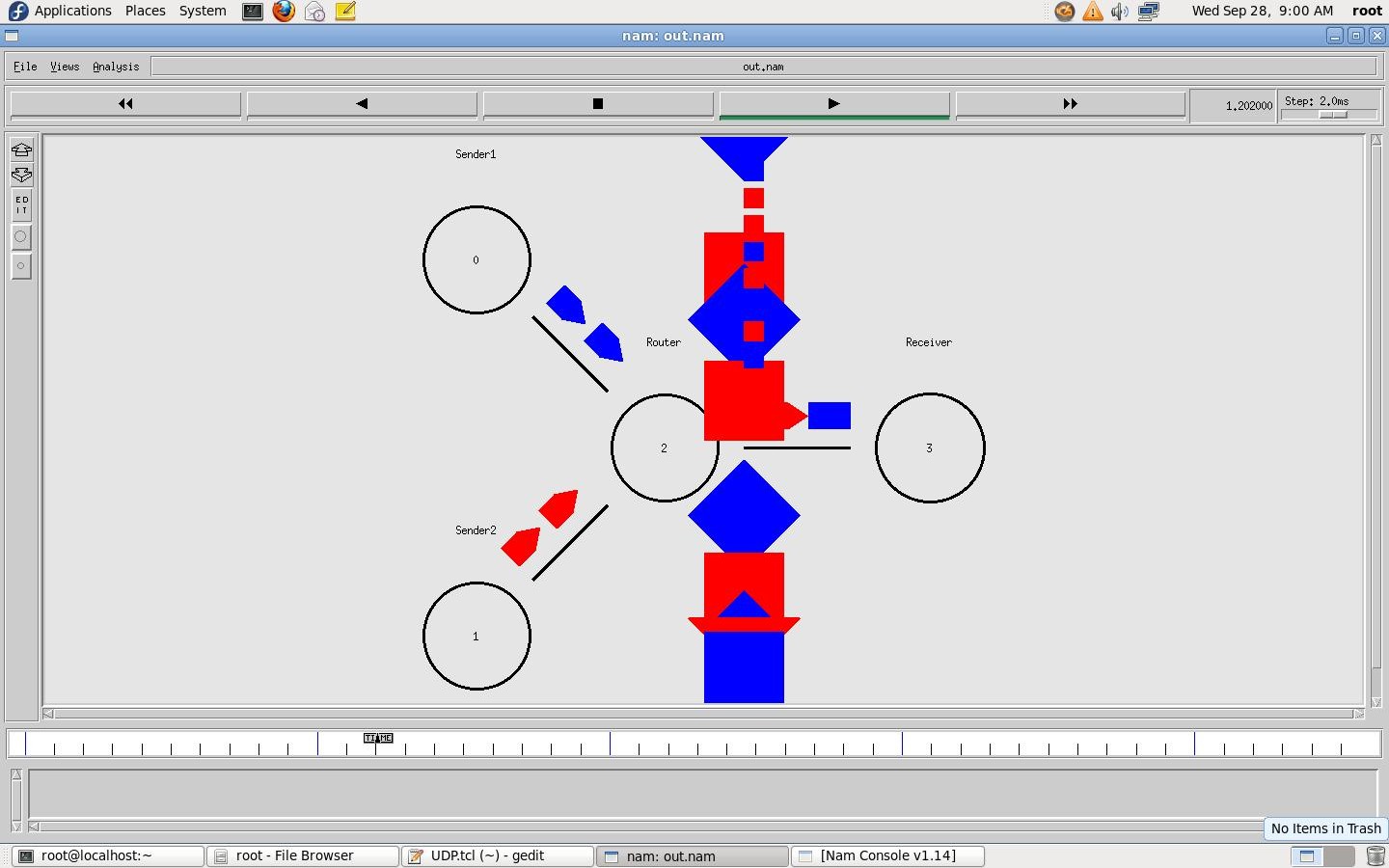
$nsat5.0"finish"

#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 Study of TCP Performance

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

setns[newSimulator]

#Open trace files

setf[opendroptail-queue-out.trw]

$nstrace-all$f

#Open the nam trace file

setnf[opendroptail-queue-out.namw]

$nsnamtrace-all$nf

#s1,s2ands3actassources. set s1 [$ns node]

set s2 [$ns node]sets3 [$ns node]

#G acts as a gatewaysetG[$ns node]

#r acts as a receiversetr[$ns node]

#Define different colors for dataflows

$nscolor1red

$nscolor2SeaGreen

$nscolor3blue

#Create links between the nodes

$nsduplex-link$s1$G6Mb10msDropTail

$nsduplex-link$s2$G6Mb10msDropTail

$nsduplex-link$s3$G6Mb10msDropTail

$nsduplex-link$G$r3Mb10msDropTail

#Define the layout of the nodes

$nsduplex-link-op$s1$Gorientright-up

$nsduplex-link-op$s2$Gorientright

$nsduplex-link-op$s3$Gorientright-down

$nsduplex-link-op$G$rorientright

#Define the queue size for the link between node G and r

$nsqueue-limit$G$r5

#Monitorthequeuesforlinksvertically

$nsduplex-link-op$s1$GqueuePos0.5

$nsduplex-link-op$s2$GqueuePos0.5

$nsduplex-link-op$s3$GqueuePos0.5

$nsduplex-link-op$G$rqueuePos 0.5

#Createa TCP agent and attach it to node s1 settcp1 [new Agent/TCP/Reno]

$ns attach-agent$s1$tcp1

$tcp1setwindow\_8

$tcp1set fid\_ 1

#Createa TCP agent and attach it to node s2 set tcp2 [new Agent/TCP/Reno]

$ns attach-agent$s2$tcp2

$tcp2setwindow\_8

$tcp2set fid\_ 2

#Createa TCP agent and attach it to node s3 set tcp3 [new Agent/TCP/Reno]

$nsattach-agent$s3$tcp3

$tcp3setwindow\_4

$tcp3set fid\_ 3

#Create TCP sink agents and attach them to node r set sink1 [new Agent/TCPSink]

setsink2[newAgent/TCPSink] setsink3[newAgent/TCPSink]

$nsattach-agent$r$sink1

$nsattach-agent$r$sink2

$nsattach-agent$r$sink3

#Connect the traffic sources with the traffic sinks

$nsconnect$tcp1$sink1

$nsconnect$tcp2$sink2

$nsconnect$tcp3$sink3

#Create FTP applications and attach them to agents set ftp1 [new Application/FTP]

$ftp1attach-agent$tcp1

setftp2[newApplication/FTP]

$ftp2attach-agent$tcp2

setftp3[newApplication/FTP]

$ftp3attach-agent$tcp3

#Definea'finish'procedure proc finish {} {

globalns

$nsflush-trace

puts"runningnam."

execnam-adroptail-queue-out.nam& exit 0

**}**

#Define label for nodes

$nsat0.0"$s1labelSender1"

$nsat0.0"$s2labelSender2"

$nsat0.0"$s3labelSender3"

$nsat0.0"$GlabelGateway"

$nsat0.0"$rlabelReceiver"

#Schedule ftp events

$nsat0.1"$ftp1start"

$nsat0.1"$ftp2start"

$nsat0.1"$ftp3start"

$nsat5.0"$ftp1stop"

$nsat5.0"$ftp2stop"

$nsat5.0"$ftp3stop"

#Call finish procedure after 5seconds of simulation time

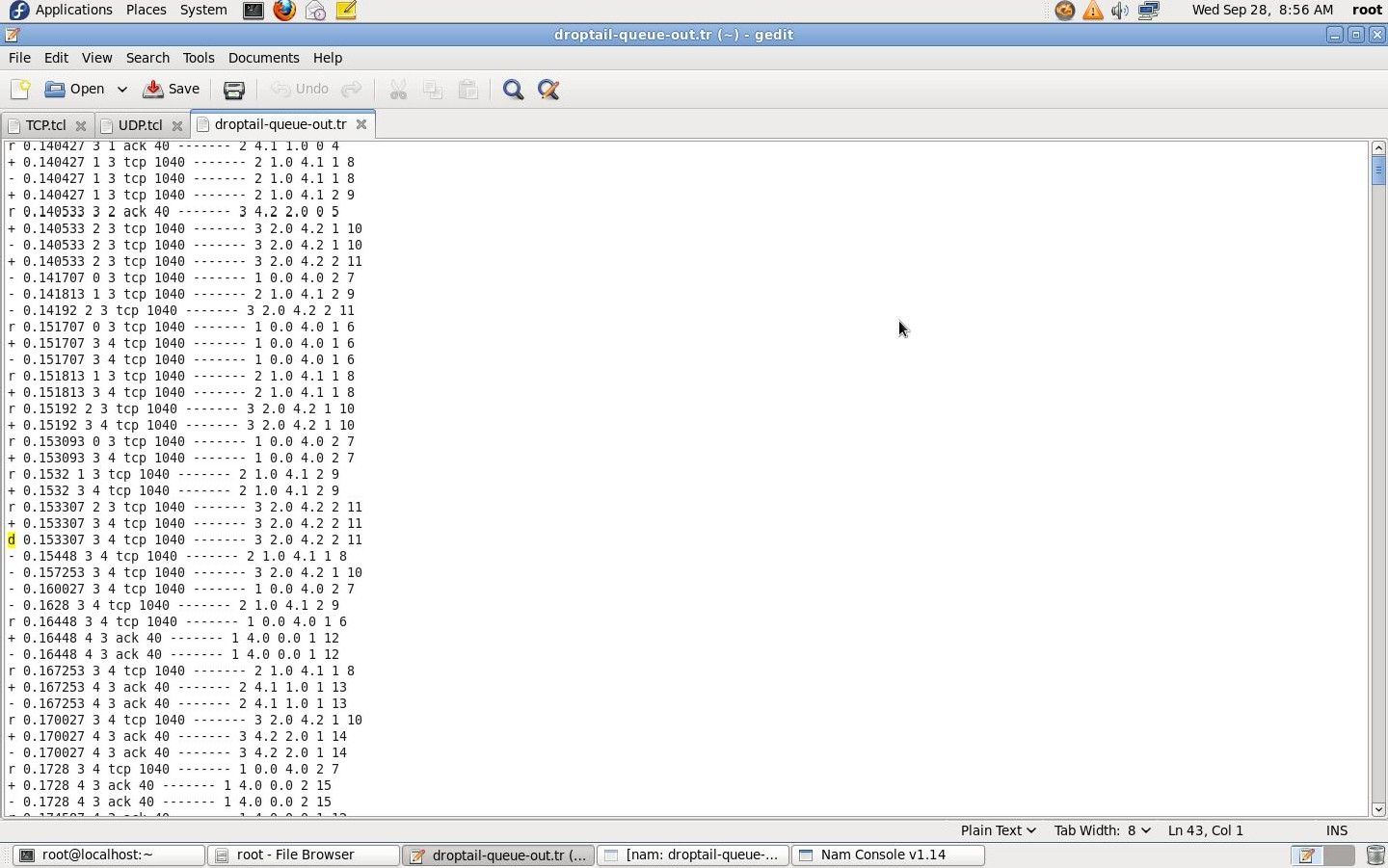
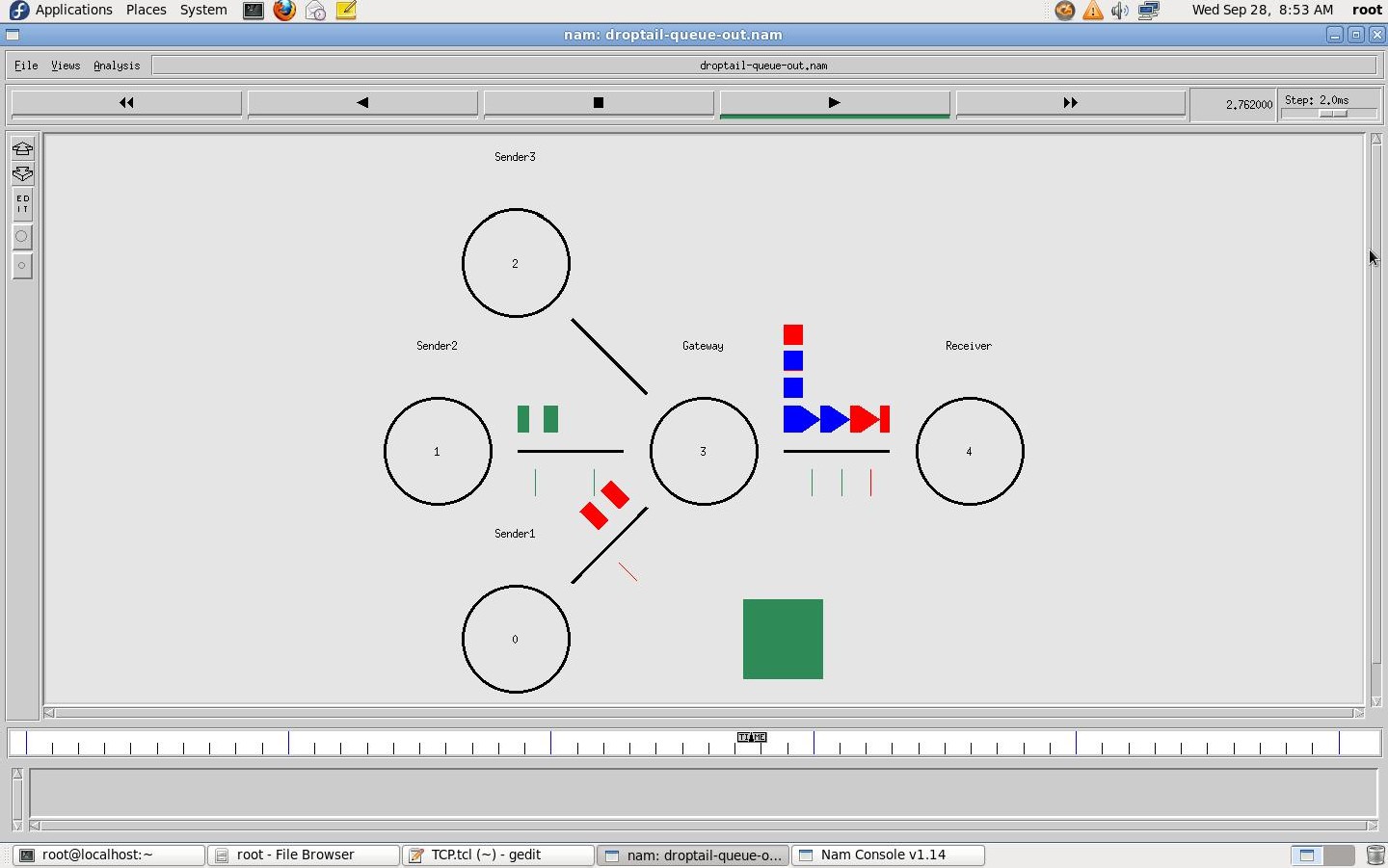
$nsat5.25"finish"

#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 Distance Vector Routing Protocol

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

$nsrtproto DV

#Open the nam trace file set nf [open out.nam w]

$nsnamtrace-all$nf

#Open tracefile

setnt[opentrace.trw]

$nstrace-all$nt

#Define 'finish' procedureprocfinish{}

**{**

globalnsnf

$ns flush-trace #Closethetrace file close $nf

#Execute nam on the trace fileexecnam-aout.nam& exit 0

**}**

#Create8nodes 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]

#Specify link characterestics

$nsduplex-link$n1$n21Mb10msDropTail

$nsduplex-link$n2$n31Mb10msDropTail

$nsduplex-link$n3$n41Mb10msDropTail

$nsduplex-link$n4$n51Mb10msDropTail

$nsduplex-link$n5$n61Mb10msDropTail

$nsduplex-link$n6$n71Mb10msDropTail

$nsduplex-link$n7$n81Mb10msDropTail

$nsduplex-link$n8$n11Mb10msDropTail

#specify layout as aoctagon

$nsduplex-link-op$n1$n2orientleft-up

$nsduplex-link-op$n2$n3orient up

$nsduplex-link-op$n3$n4orientright-up

$nsduplex-link-op$n4$n5orientright

$nsduplex-link-op$n5$n6orientright-down

$nsduplex-link-op$n6$n7orientdown

$nsduplex-link-op$n7$n8orientleft-down

$nsduplex-link-op$n8$n1orientleft

#Createa UDP agent and attach it to node n1 set udp0 [new Agent/UDP]

$nsattach-agent$n1$udp0

#Createa CBR traffic source and attach it to udp0 set cbr0 [new Application/Traffic/CBR]

$cbr0setpacketSize\_500

$cbr0setinterval\_0.005

$cbr0attach-agent$udp0

#Createa Null agent(a traffic sink)and attach it to node n4 set null 0 [new Agent/Null]

$nsattach-agent$n4$null0

#Connect the traffic source with the traffic sink

$nsconnect$udp0$null0

#Schedule events for the CBR agent and the network dynamics

$nsat0.0 "$n1labelSource"

$nsat0.0 "$n4labelDestination"

$nsat0.5"$cbr0start"

$nsrtmodel-at1.0down$n3$n4

$nsrtmodel-at2.0up$n3$n4

$nsat4.5"$cbr0stop"

#Call the finish procedure after 5seconds of simulation time

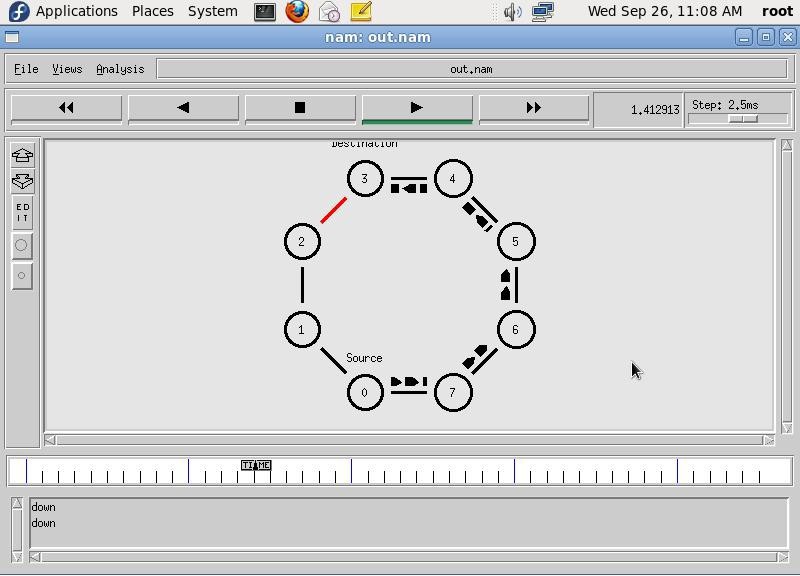
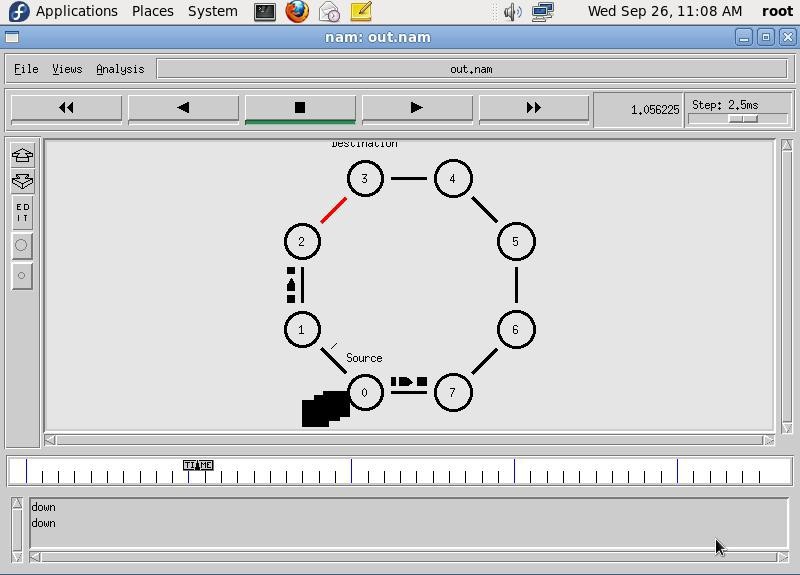
$nsat5.0"finish"

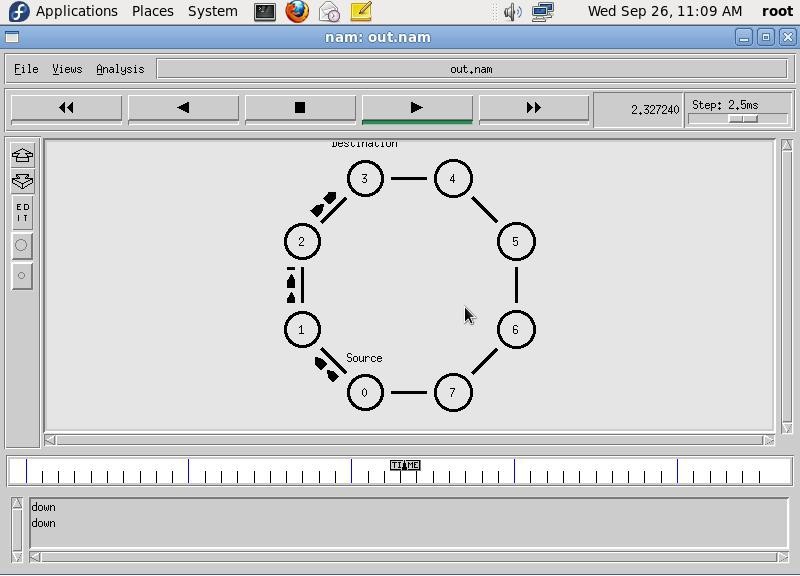
#Run the simulation

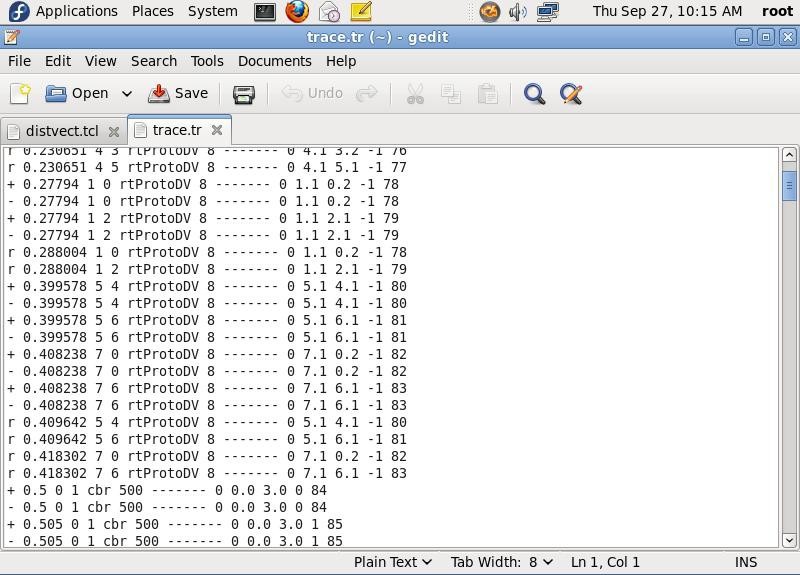
$ns run

OUTPUT:

$ ns distvect.tcl







Result

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

# Exp.No.9b Link State Routing Protocol

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

setns[newSimulator] set nr [open thro.tr w]

$nstrace-all$nr

setnf[openthro.namw]

$nsnamtrace-all

$nfprocfinish{}

**{**

globalnsnrnf

$nsflush-trace close$nf close $nr

exec nam thro.nam&exit 0

**}**

for{seti0}{$i<12}{incri1}{ set n($i) [$ns node]}

for{seti0}{$i<8}{incri}{

$nsduplex-link$n($i)$n([expr$i+1])1Mb10msDropTail}

$nsduplex-link$n(0)$n(8)1Mb10msDropTail

$nsduplex-link$n(1)$n(10)1Mb10msDropTail

$nsduplex-link$n(0)$n(9)1Mb10msDropTail

$nsduplex-link$n(9)$n(11)1Mb10msDropTail

$nsduplex-link$n(10)$n(11)1Mb10msDropTail

$nsduplex-link$n(11)$n(5)1Mb10msDropTail

setudp0[newAgent/UDP]

$nsattach-agent$n(0)$udp0

setcbr0[newApplication/Traffic/CBR]

$cbr0setpacketSize\_500

$cbr0setinterval\_0.005

$cbr0attach-agent$udp0 set null0 [new Agent/Null]

$nsattach-agent$n(5)$null0

$nsconnect$udp0$null0

setudp1[newAgent/UDP]

$nsattach-agent$n(1)$udp1

setcbr1[newApplication/Traffic/CBR]

$cbr1setpacketSize\_500

$cbr1setinterval\_0.005

$cbr1attach-agent$udp1 set null0 [new Agent/Null]

$nsattach-agent$n(5)$null0

$nsconnect$udp1$null0

$nsrtproto LS

$nsrtmodel-at10.0down$n(11)$n(5)

$nsrtmodel-at15.0down$n(7)$n(6)

$nsrtmodel-at30.0up$n(11)$n(5)

$nsrtmodel-at20.0up$n(7) $n(6)

$udp0setfid\_1

$udp1setfid\_2

$nscolor1Red

$nscolor2Green

$nsat1.0"$cbr0start"

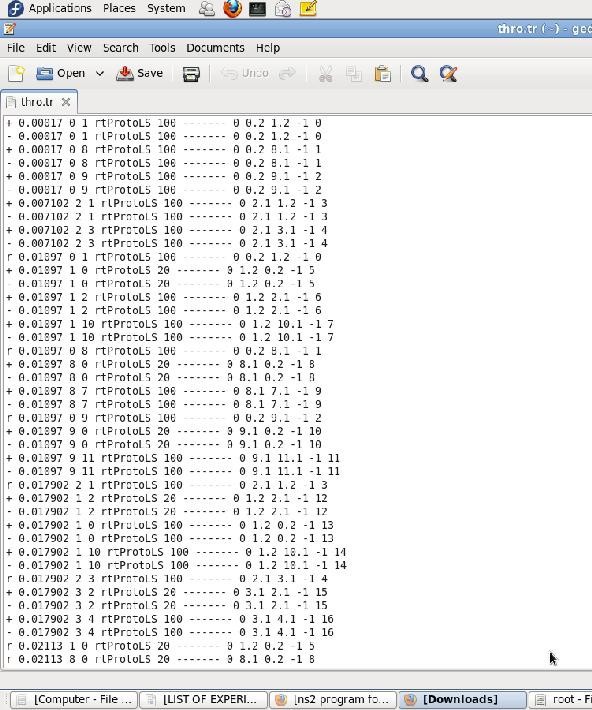
$nsat2.0"$cbr1start"

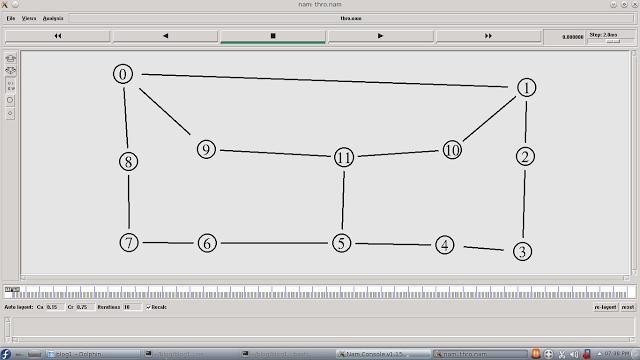
$nsat45"finish"

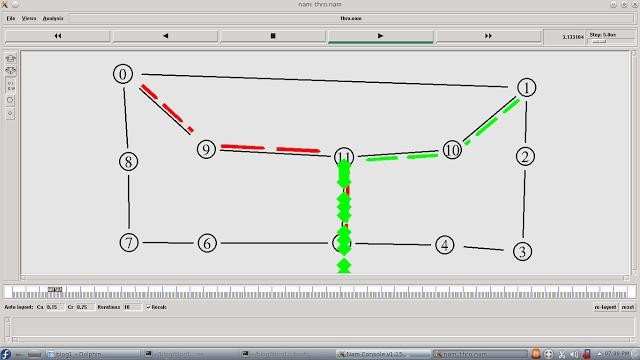
$ns run

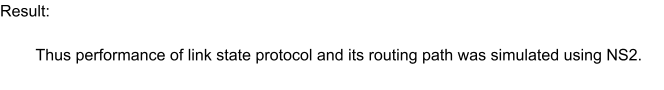
OUTPUT:

$ ns ls.tcl









# Exp#10 CRC Error Detection

Date:

Aim

To detect whether the given data is corrupted or not using CRC method.

Algorithm

1. Read number of data bits.
2. Read the data bit-by-bit
3. Read number of divisor bits
4. Enter the divisor bit-by-bit
5. Append zeroes to the message
6. Generate remainder by using XOR division
7. Subtract remainder from message using XOR
8. Display the CRC code word
9. Accept transmitted message as receiver side data
10. Perform polynomial division using XOR
11. If remainder is zero then display“No error”else display“Error”
12. Stop

Program

import java.io.\*;

class crcgen {

public static void main(String args[]) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int[] data;

int[] div;

int[] divisor;

int[] rem;

int[] crc;

int data\_bits, divisor\_bits, tot\_length;

// Input data bits

System.out.println("Enter number of data bits:");

data\_bits = Integer.parseInt(br.readLine());

data = new int[data\_bits];

System.out.println("Enter data bits:");

for (int i = 0; i < data\_bits; i++)

data[i] = Integer.parseInt(br.readLine());

// Input divisor (generator polynomial)

System.out.println("Enter number of bits in divisor:");

divisor\_bits = Integer.parseInt(br.readLine());

divisor = new int[divisor\_bits];

System.out.println("Enter Divisor bits:");

for (int i = 0; i < divisor\_bits; i++)

divisor[i] = Integer.parseInt(br.readLine());

// Prepare for division: data bits + (divisor\_bits - 1) zeros

tot\_length = data\_bits + divisor\_bits - 1;

div = new int[tot\_length];

rem = new int[tot\_length];

crc = new int[tot\_length];

// Copy data bits into div array

for (int i = 0; i < data.length; i++)

div[i] = data[i];

System.out.print("Dividend (after appending 0's): ");

for (int i = 0; i < div.length; i++)

System.out.print(div[i]);

System.out.println();

// Copy div into rem for division

for (int j = 0; j < div.length; j++)

rem[j] = div[j];

rem = divide(div, divisor, rem);

// Append remainder to data

for (int i = 0; i < div.length; i++) {

crc[i] = (div[i] ^ rem[i]);

}

System.out.println("CRC code:");

for (int i = 0; i < crc.length; i++)

System.out.print(crc[i]);

// Error Detection

System.out.println("\nEnter CRC code of " + tot\_length + " bits:");

for (int i = 0; i < crc.length; i++)

crc[i] = Integer.parseInt(br.readLine());

for (int j = 0; j < crc.length; j++)

rem[j] = crc[j];

rem = divide(crc, divisor, rem);

boolean error = false;

for (int i = 0; i < rem.length; i++) {

if (rem[i] != 0) {

System.out.println("Error");

error = true;

break;

}

}

if (!error)

System.out.println("No Error");

}

static int[] divide(int div[], int divisor[], int rem[]) {

int cur = 0;

while (true) {

for (int i = 0; i < divisor.length; i++)

rem[cur + i] = (rem[cur + i] ^ divisor[i]);

while (cur < rem.length && rem[cur] == 0)

cur++;

if ((rem.length - cur) < divisor.length)

break;

}

return rem;

}

}

**Output:**

Enter number of data bits:

4

Enter data bits:

1

0

0

1

Enter number of bits in divisor:

3

Enter Divisor bits:

1

0

1

Dividend (after appending 0's): 100100

CRC code: 100111

Enter CRC code of 6 bits:

1

0

0

1

1

1

No Error

Result

Thus error detection is done using cyclic redundancy check method.