NITTE EDUCATION TRUST

NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY
An Autonomous Institution Approved by UGC/AICTE/Govt. of Karnataka
Accredited by NBA (Tier-I) and NAAC 'A+' Grade
Affiliated to Visvesvaraya Technological University, Belagavi
Post Box No. 6429, Yelahanka, Bengaluru-560064, Karnataka, INDIA

KNOWLEDGE • CHARACTER • UNITY

Department of Electronics and Communication Engineering

KNOWLEDGE • CHARACTER • UNITY

Data Communication Networks Laboratory Manual

Program: B.E.
Course: ECE
Semester: 7^{th}
Subject Code: 18ECL77

SEMESTER: VII DATA COMMUNICATION NETWORK LAB			
Course Code	**18ECL77**	**Credits**	**01**
Hours/Week(L-T-P-S)	**0-0-2**	**CIE Marks**	**50**
Total Hours	**26 (P)**	**SEE Marks**	**50**

PRE-REQUISITES

Knowledge of C Programming
Basic Networking

COURSE OUTCOMES

Student will be able to:

1. Apply the principles of computer networks.
2. Analyze the functionality of layered network architecture.
3. Apply different protocols to design and implement in wired/wireless networks.
4. Compare different routing algorithms.
5. Analyze and implement error control coding techniques.

COURSE CONTENTS

1. Simulate a three nodes point-to-point network with duplex links between them. Set the queue size vary the bandwidth to find the number of packets dropped.
2. Simulate a four node point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP n1-n3. Apply relevant applications over TCP and UDP agents to determine the number of packets sent by TCP/UDP
3. Simulate an Ethernet LAN using N nodes .Set multiple traffic nodes and determine collision across different nodes.
4. Simulate an Ethernet LAN using N-nodes(6-10), change bandwidth and compare the throughput
5. Simulate simple BSS and with transmitting nodes in wire-less LAN. Determine the performance with respect to transmission of packets.
6. Simulate transmission of ping messages over a network topology and capture the Round Trip Time.
7. Simulate a 6 node network to implement Dynamic Routing algorithm and verify its functionality.
8. Implement a method of cyclic data transmission using UDP protocol.
9. Implement using C, the error detecting code CRC for 16 bits.
10. Implement using C ,Hamming Code generation for error detection and correction
11. Simulate a wireless network to test Destination-Sequenced Distance-Vector Routing (DSDV) protocol.
12. Simulate a 7 node network to verify Link State Routing protocol.

TEXT BOOK

1. Behrouzs Forouzan, “Data communication and networking”, TMH, 4th Edition, 2006.

REFERENCE BOOKS
1. Bhushsan Trivedi “Data Communication and Network ,” Oxford Higher Education, 1st edition , 2016

TEACHING METHODOLOGY
• **Black board teaching** • **PowerPoint presentations (If needed)** • **Regular review of students by asking questions based on topics covered in the Lab**

COURSE ASSESSEMENT METHOD

CIE:

1. Regular review of the students work in the lab and recording the work done on a day to day basis in Record and observation.
2. Conduction Exam for the students to assess the quality of the work done in the lab. We do modify thenetwork setup, calculation to make sure students have understood the tool.
3. Viva voce examination to analyze the student’s involvement in the experiment.
4. Record and Observation carries 30 marks
5. Execution carries 15 marks
6. Viva Voce carries 5 marks
7. Total CIE is foe 50 marks.

SEE:

1. One question is set to related to each experiment. We do modify the network setup, calculation to makesure students have understood the tool.
2. Write up is awarded 8 marks
3. Execution of the program is awarded 35marks.
4. Viva voce examination for the students is awarded with 7 marks
5. Total marks for SEE is 50 marks

CO-PO-PSO MAPPING

CO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3	BT
CO1	3	3			3		2	1	1	1		3		3		3
CO2	3	3			3		2	1	1	1		3		3		3
CO3	3	3		1	3		2	1	1	1		3		3		3
CO4	3	3	2	1	3		2	1	1	1		3		3		2
CO5	3	3	2		3		2	1	1	1		**3**		3		3

Introduction to Network Simulator 2 (NS2)

What is Simulation?

"The process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and/or evaluating various strategies for the operation of the system." Network Simulator (NS2) is an open-source event-driven simulator designed specifically for research in computer communication networks. A computer network is usually defined as a collection of computers interconnected for gathering, processing, and distributing information. The Internet is a good example of computer networks. In fact, it is a network of networks, within which, tens of thousands of networks interconnect millions of computers worldwide.

Advantages of NS2

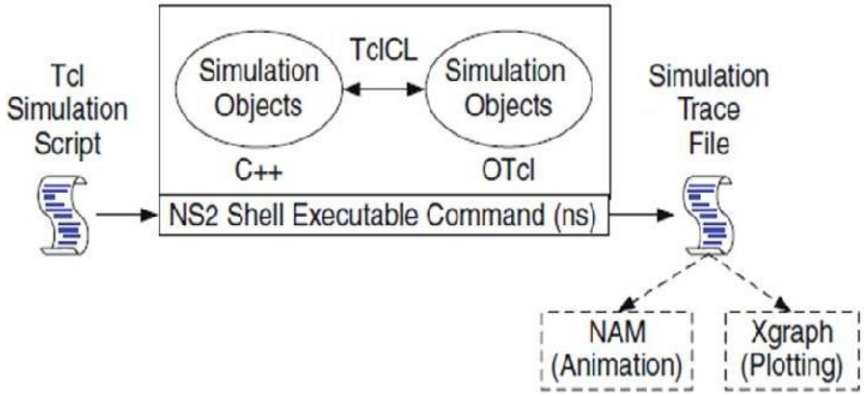
- Cost effective
- Flexible
- Easier to analyze
- Provide substantial support to simulate bunch of protocols like TCP, UDP, FTP, and HTTP

Programming Language: C++

Scripting Language: Tool Kit Command Language (TCL)

Architecture of NS2

The network simulator NS is a discrete event network simulator developed at UC Berkeley that focuses on the simulation of IP networks on the packet level. The NS project (the project that drives the development of NS) is now part of the Virtual InterNetwork Testbed (VINT) project, that develops tools for network simulation research. Researchers have used NS to develop and investigate protocols such as TCP and UDP, router queuing policies (RED, ECN, CBQ), Multicast transport, Multimedia and more .



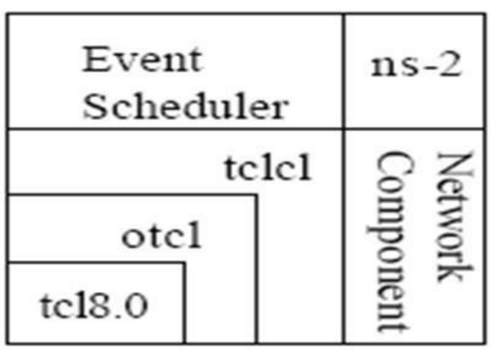
Simplified User View of NS

NS is basically an Object-oriented Tcl (Otcl) script interpreter with network simulation object libraries. NS has a simulation event scheduler, network component object libraries and network setup (plumbing) modul libraries. To use NS for setting up and running a network simulation, a user writes a simulation program in Otcl script language. Such an OTcl script initiates an event scheduler, sets up the network topology and tells traffic sources when to start and stop transmitting packets through the event scheduler.

Architectural Overview

The NS-2 architecture is composed of five parts:

- Event scheduler
- Network components
- Tclcl
- OTcl library
- Tcl 8.0 scipt language



The above figure shows a graphical overview of the NS-2 architecture. A user can be thought of standing at the left bottom corner, designing and running simulations in Tcl using the simulator objects in the OTcl library." The event schedulers and most of the network components are implemented in C++ because of efficiency reasons. These are available to OTcl through an OTcl linkage that is implemented using tclcl. These five components together make up NS, which is an object-oriented extended Tcl interpreter with network simulator libraries. NS models all network elements through a class hierarchy. For example, Agent is a class TCP and UDP under it. To drive the execution of the simulation, to process and schedule simulation events, NS makes use of the concept of discrete event schedulers. In NS, network components that simulate packet-handling delay or that need timers use event schedulers.

Experiment 1

Aim: "Simulate a three nodes' point-to-point network with duplex links between them. Set the queue size vary the bandwidth and find the number of packets dropped"

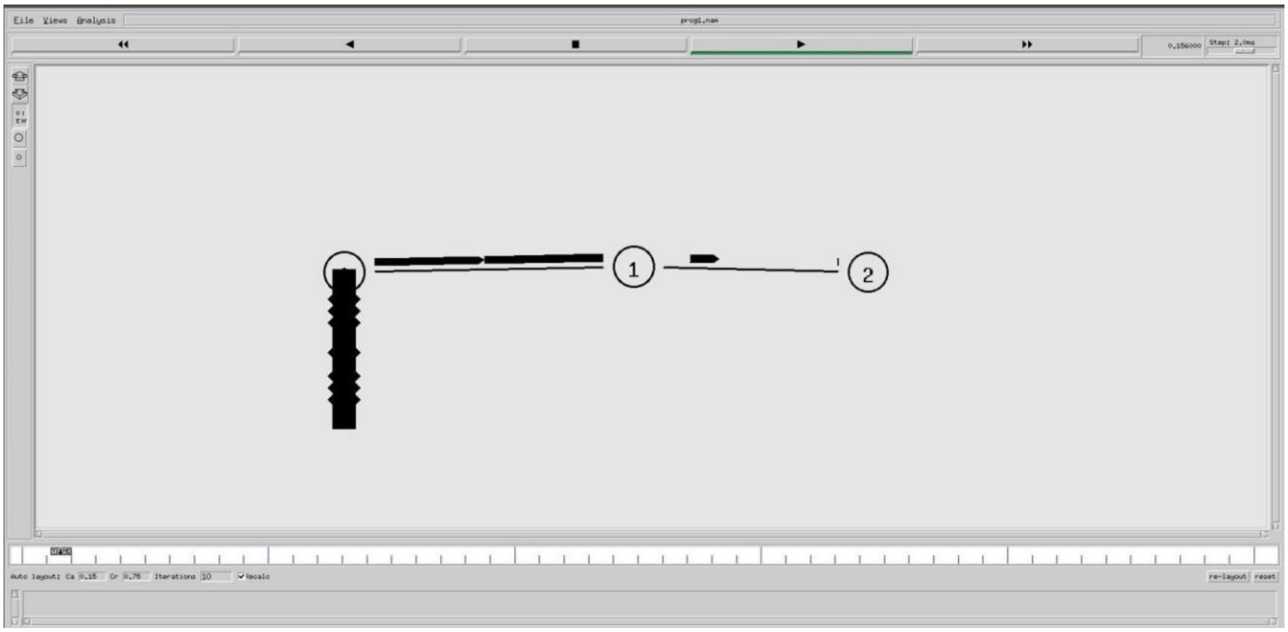
Tool Commanding Language (TCL) Code

```
#Create Simulator
set ns [new Simulator]
#Open Trace file and NAM file
set ntrace [open prog1.tr w]
$ns trace-all $ntrace
set namfile [open prog1.nam w]
$ns namtrace-all $namfile
#Finish Procedure
proc Finish {} {
global ns ntrace namfile
#Dump all the trace data and close the files
$ns flush-trace
close $ntrace
close $namfile
#/Execute the nam animation file
exec nam prog1.nam &
exec echo "The number of packets dropped are:" &
exec grep -c "^d" prog1.tr &
exit 0
}
#Create 3 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
#Create Links between nodes
#You need to modify the bandwidth to observe the variation in packet drop
$ns duplex-link $n0 $n1 0.2Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
#Set Queue Size
#You can modify the queue length as well to observe the variation in packet drop
$ns queue-limit $n0 $n1 10
$ns queue-limit $n1 $n2 10
#Set up a Transport layer connection.
```

```
set udp [new Agent/UDP]
$ns attach-agent $n0 $udp
set null [new Agent/Null]
$ns attach-agent $n2 $null
$ns connect $udp $null
#Set up an Application layer Traffic
set cbr0 [new Application/Traffic/CBR]
#$cbr0 set type_ CBR
#$cbr0 set packetSize_ 100
#$cbr0 set rate_ 1Mb
#$cbr0 set random_ false
$cbr0 attach-agent $udp
#Schedule Events
$ns at 0.0 "$cbr0 start"
$ns at 5.0 "Finish"
#Run the Simulation
$ns run
```

Terminal and NAM Outputs

```
try@try:~/NS-Programs/P1$ ns p1.tcl
The number of packets dropped are:
try@try:~/NS-Programs/P1$ 729
```



Experiment 2

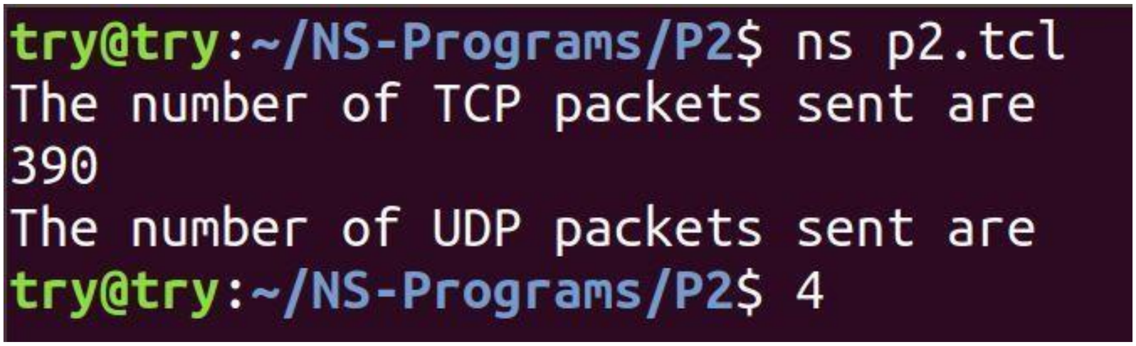
Aim: "Simulate a four-node point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP n1-n3. Apply relevant applications over TCP and UDP agents to determine the number of packets sent by TCP/UDP"

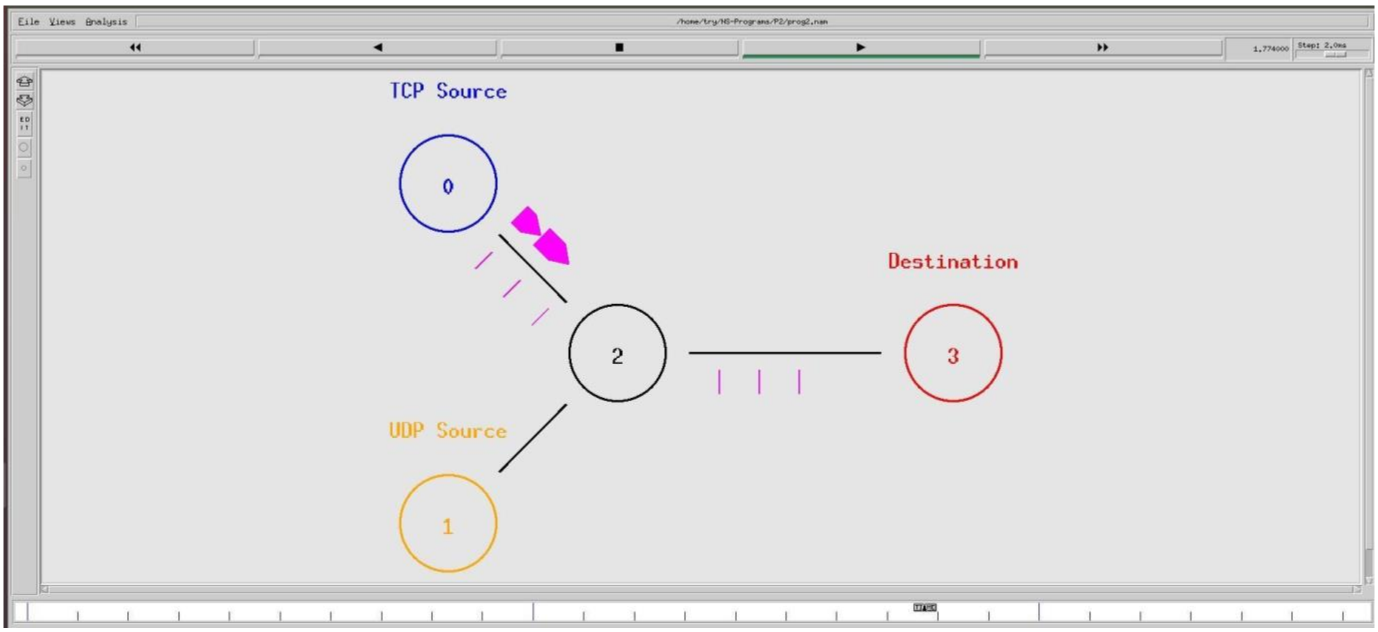
TCL Code

```
set ns [ new Simulator]
set ntrace [open prog2.tr w]
$ns trace-all $ntrace
set namfile [open prog2.nam w]
$ns namtrace-all $namfile
proc Finish {} {
global ns ntrace namfile
$ns flush-trace
close $ntrace
close $namfile
exec nam prog2.nam &
exec echo "The number of TCP packets sent are" &
exec grep "^+" prog2.tr | cut -d " " -f 5 | grep -c "tcp" &
exec echo "The number of UDP packets sent are" &
exec grep "^+" prog2.tr | cut -d " " -f 5 | grep -c "cbr" &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 2Mb 20ms DropTail
########### EXTRA CODE##########################
$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

$n0 label "TCP Source"
$n1 label "UDP Source"
```

```
$n3 label "Destination"
$n0 color blue
$n1 color orange
$n3 color red
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0
set ftp0 [new Application/FTP]
$ftp0 set type_ FTP
$ftp0 attach-agent $tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set type_ CBR
$cbr0 set packetSize_ 1000
$cbr0 set rate_ 0.01Mb
$cbr0 set random_ false
$cbr0 attach-agent $udp0
$ns color 1 magenta
$ns color 2 green
$tcp0 set class_ 1
$udp0 set class_ 2
$ns at 0.1 "$cbr0 start"
$ns at 1.5 "$ftp0 start"
$ns at 1.0 "$cbr0 stop"
$ns at 2.5 "$ftp0 stop"
$ns at 5.0 "Finish"
$ns run
```

Terminal and NAM Outputs





Experiment No 3

Aim: "Simulate an Ethernet LAN using N nodes. Set multiple traffic nodes and determine collision across different nodes".

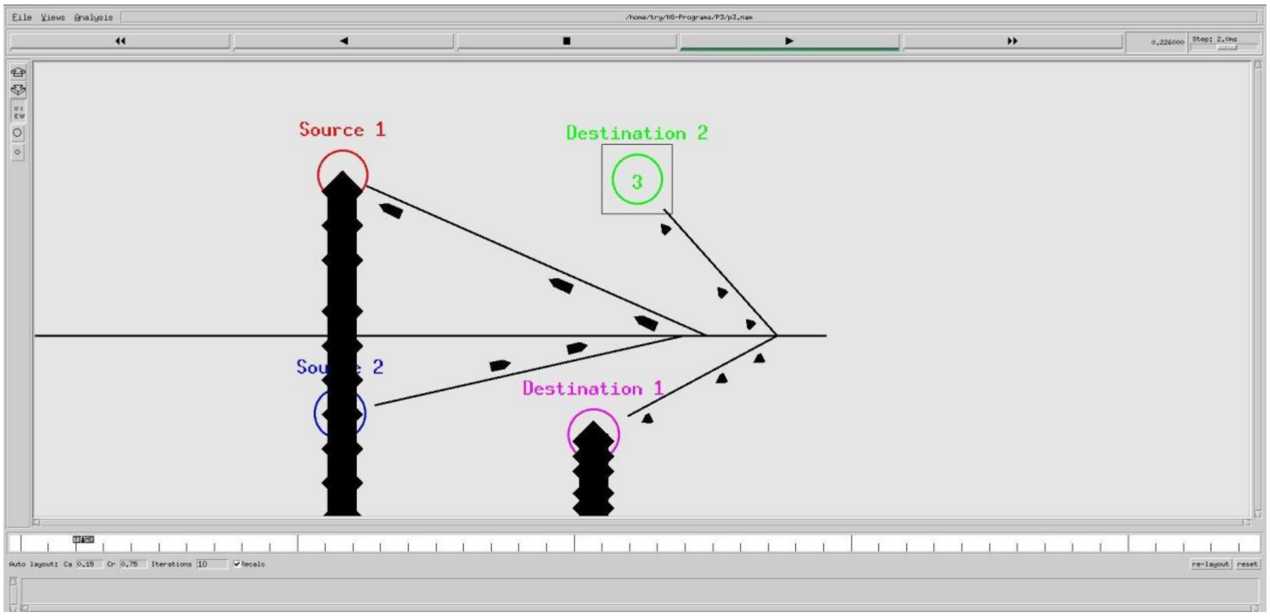
TCL Code

```
set ns [new Simulator]
set trf [open p3.tr w]
$ns trace-all $trf
set naf [open p3.nam w]
$ns namtrace-all $naf
set n0 [$ns node]
$n0 color "red"
$n0 label "Source 1"
set n1 [$ns node]
$n1 color "blue"
$n1 label "Source 2"
set n2 [$ns node]
$n2 color "magenta"
$n2 label "Destination 1"
set n3 [$ns node]
$n3 color "green"
$n3 label "Destination 2"

set lan [$ns newLan "$n0 $n1 $n2 $n3" 5Mb 10ms LL Queue/DropTail Mac/802_3]

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $sink
$ns connect $tcp $sink
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
```

```
proc finish {} {
global ns naf trf
$ns flush-trace
exec nam p3.nam &
close $trf
close $naf
exec echo "The number of packet drops due to collision are" &
exec grep -c "^d" p3.tr &
exit 0
}

$ns at 0.1 "$cbr start"
$ns at 2.0 "$ftp start"
$ns at 1.9 "$cbr stop"
$ns at 4.3 "$ftp stop"
$ns at 6.0 "finish"
$ns run
```

Terminal and NAM Outputs

```
try@try:~/NS-Programs/P3$ ns p3.tcl
warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

The number of packet drops due to collision are
try@try:~/NS-Programs/P3$ 3006
```



Experiment 4

Aim: "Simulate an Ethernet LAN using N-nodes (6-10), change bandwidth and compare the throughput"

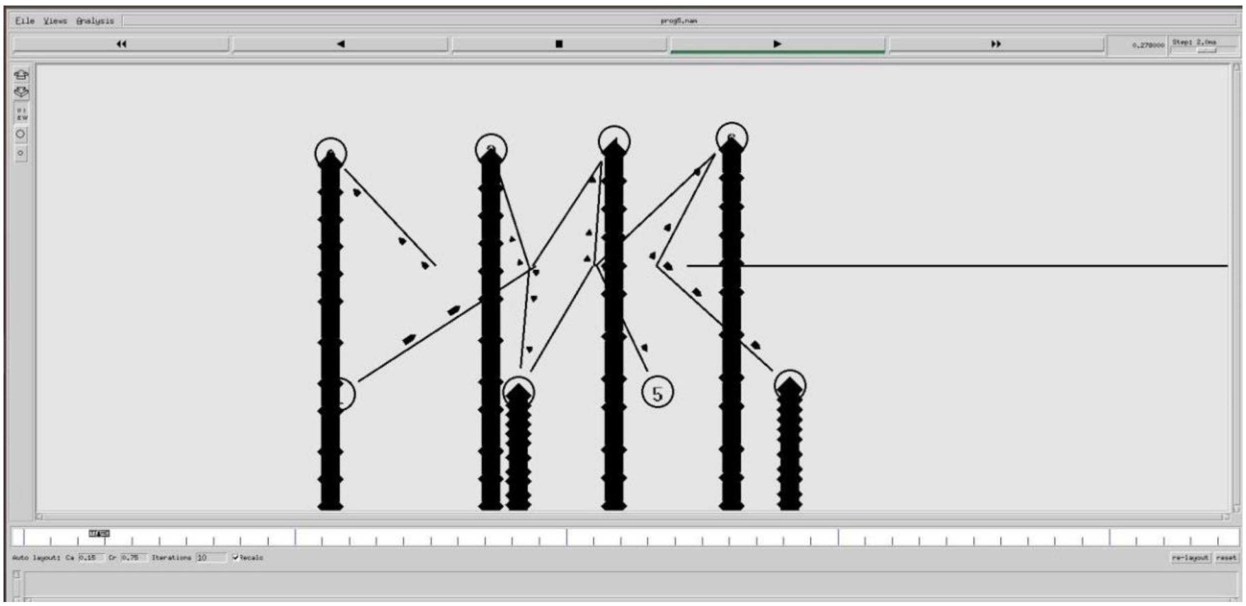
TCL Code

```
set ns [new Simulator]
set trf [open prog5.tr w]
$ns trace-all $trf
set naf [open prog5.nam w]
$ns namtrace-all $naf

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 lan [$ns newLan "$n0 $n1 $n2 $n3 $n4 $n5 $n6 $n7" 5Mb 10ms LL Queue/DropTail Channel]

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n7 $sink
$ns connect $tcp $sink
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null

proc finish {} {
global ns naf trf
$ns flush-trace
exec nam prog5.nam &
close $trf
close $naf
```

```
set tcpsize [ exec grep "^r" prog5.tr | grep "tcp" | tail -n 1 | cut -d " " -f 6]
set numtcp [ exec grep "^r" prog5.tr | grep -c "tcp"]
set tcptime 2.3
set udpsize [ exec grep "^r" prog5.tr | grep "cbr" | tail -n 1 | cut -d " " -f 6]
set numudp [ exec grep "^r" prog5.tr | grep -c "cbr"]
set udptime 4.0

puts "The throughput of FTP is"
puts "[ expr ($numtcp*$tcpsize)/$tcptime] bytes per second"
puts "The throughput of CBR is"
puts "[ expr ($numudp*$udpsize)/$udptime] bytes per second"
exit 0
}

$ns at 0.1 "$cbr start"
$ns at 2.0 "$ftp start"
$ns at 1.9 "$cbr stop"
$ns at 4.3 "$ftp stop"
$ns at 6.0 "finish"
$ns run
```

Terminal and NAM Outputs

```
try@try:~/NS-Programs/P4$ ns p4.tcl
warning: no class variable LanRouter::debug_

        see tcl-object.tcl in tclcl for info about this warning.

The throughput of FTP is
231060.86956521741 bytes per second
The throughput of CBR is
25252.5 bytes per second
try@try:~/NS-Programs/P4$
```



Experiment 5

Aim: "Simulate simple BSS and with transmitting nodes in wire-less LAN. Determine the performance with respect to transmission of packets".

TCL Code

```
# Create a NS simulator object
set ns [new Simulator]
#setup trace support by opening file p5.tr and call the procedure trace-all
set tf [open p5.tr w]
$ns trace-all $tf
#create a topology object that keeps track of movements of mobile nodes
#within the topological boundary.
set topo [new Topography]
$topo load_flatgrid 1000 1000
set nf [open p5.nam w]
$ns namtrace-all-wireless $nf 1000 1000
# creating a wireless node you MUST first select (configure) the node
#configuration parameters to "become" a wireless node.
#Destination-Sequenced Distance-Vector Routing (DSDV)---------------DSDV or DSR or TORA
$ns node-config -adhocRouting DSDV \
-llType LL \
-macType Mac/802_11 \
-ifqType Queue/DropTail \
-ifqLen 50 \
-phyType Phy/WirelessPhy \
-channelType Channel/WirelessChannel \
-propType Propagation/TwoRayGround \
-antType Antenna/OmniAntenna \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON
# Create god object
create-god 3
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"
$n0 set X_ 50
$n0 set Y_ 50
$n0 set Z_ 0
$n1 set X_ 100
$n1 set Y_ 100
```

```
$n1 set Z_ 0
$n2 set X_ 600
$n2 set Y_ 600
$n2 set Z_ 0
$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 100 100 25"
$ns at 0.1 "$n2 setdest 600 600 25"
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2
$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"
$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 70 70 15"
proc finish { } {
global ns nf tf
$ns flush-trace
exec nam p5.nam &
exec awk -f p5.awk p5.tr &
close $tf
exit 0
}
$ns at 250 "finish"
$ns run
```

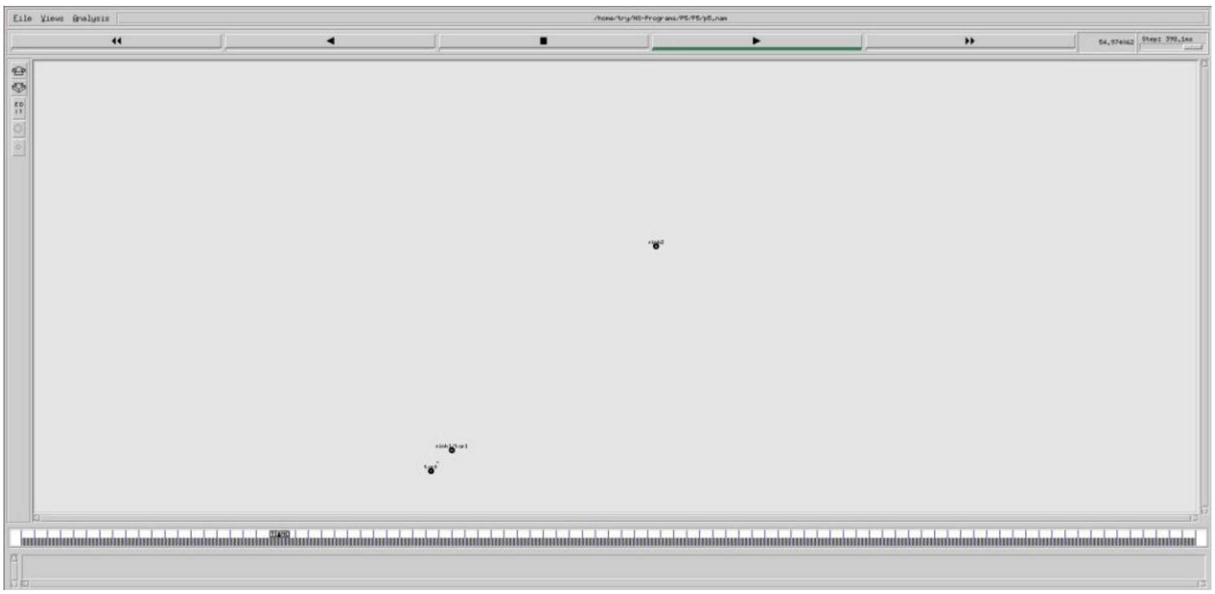
AWK Script

```
BEGIN{
count1=0
count2=0
pack1=0
pack2=0
time1=0
time2=0
```

```
}
{
if($1 == "r" && $3 == "_1_" && $4 == "AGT")
{
count1++
pack1=pack1+$8
time1=$2
}
if($1 == "r" && $3 == "_2_" && $4 =="AGT")
{
count2++
pack2=pack2+$8
time2=$2
}
}
END{
printf("\n The Throughput from n0 to n1: %f Mbps \n",
((count1*pack1*8)/(time1)));
printf("\n The Throughput from n1 to n2: %f Mbps \n",
((count2*pack2*8)/(time2)));
}
```

Terminal and NAM Outputs

```
try@try:~/NS-Programs/P5/P5$ ns p5.tcl
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
num_nodes is set 3
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!

 The Throughput from n0 to n1: 5863442244.562729 Mbps

 The Throughput from n1 to n2: 1307611834.416579 Mbps
try@try:~/NS-Programs/P5/P5$
```



Eexperiment 6

Aim: "Simulate transmission of ping messages over a network topology and capture the Round Trip Time".

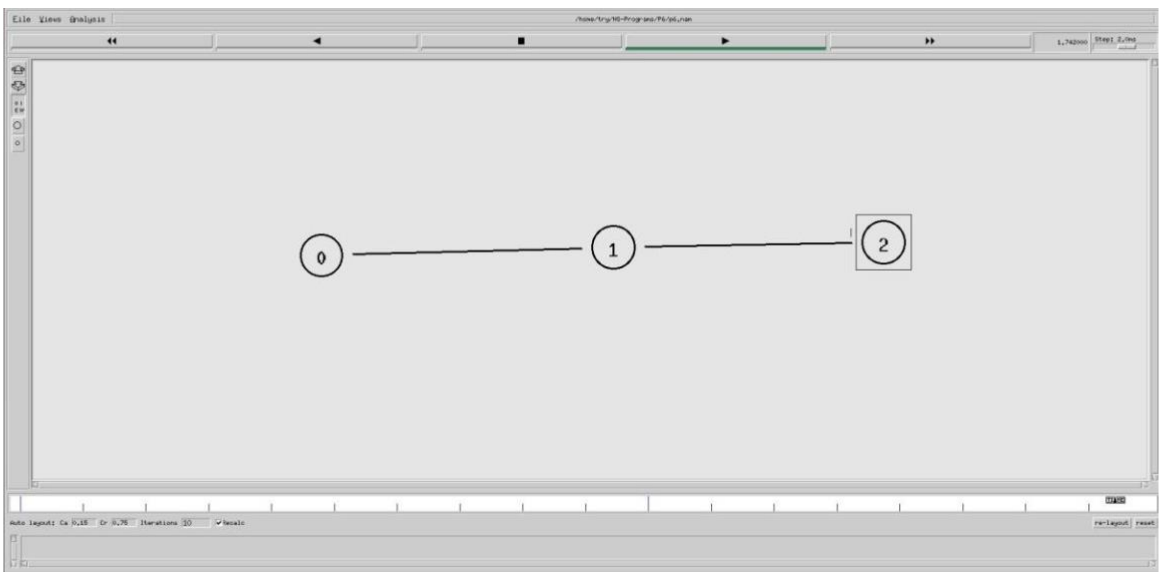
TCL Code

```
#Create Simulator
set ns [new Simulator]
#Open trace and NAM trace file
set ntrace [open p6.tr w]
$ns trace-all $ntrace
set namfile [open p6.nam w]
$ns namtrace-all $namfile
#Finish Procedure
proc Finish {} {
global ns ntrace namfile
#Dump all trace data and close the file
$ns flush-trace
close $ntrace
close $namfile
#Execute the nam animation file
exec nam p6.nam &
exit 0
}
#Create 3 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
#Define the recv function for the class 'Agent/Ping'
#instproc adds class method called "RECEIVE" to calculate RTT
Agent/Ping instproc recv {from rtt} {
#instvar adds instance variable, and brings them to the local scope
$self instvar node_
#RTT is the length of time it takes for a signal to be sent plus the length of time it takes for an
acknowledgement of that signal to be received.
puts "Node $from received ping answer from Node [$node_ id] with Round Trip Time of $rtt
ms"
}
#Create two ping agents and attach them to n(0) and n(2)
set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0
set p1 [new Agent/Ping]
```

```
$ns attach-agent $n2 $p1
$ns connect $p0 $p1
#Schedule events
$ns at 0.2 "$p0 send"
$ns at 0.4 "$p1 send"
$ns at 1.2 "$p0 send"
$ns at 1.7 "$p1 send"
$ns at 1.8 "Finish"
#Run the Simulation
$ns run
```

Terminal and NAM Outputs

```
try@try:~/NS-Programs/P6$ ls
p6.nam  P6-Old  p6.tcl  p6.tr
try@try:~/NS-Programs/P6$ ns p6.tcl
Node 2 received ping answer from Node 0 with Round Trip Time of 42.0 ms
Node 0 received ping answer from Node 2 with Round Trip Time of 42.0 ms
Node 2 received ping answer from Node 0 with Round Trip Time of 42.0 ms
Node 0 received ping answer from Node 2 with Round Trip Time of 42.0 ms
try@try:~/NS-Programs/P6$
```



Experiment 7

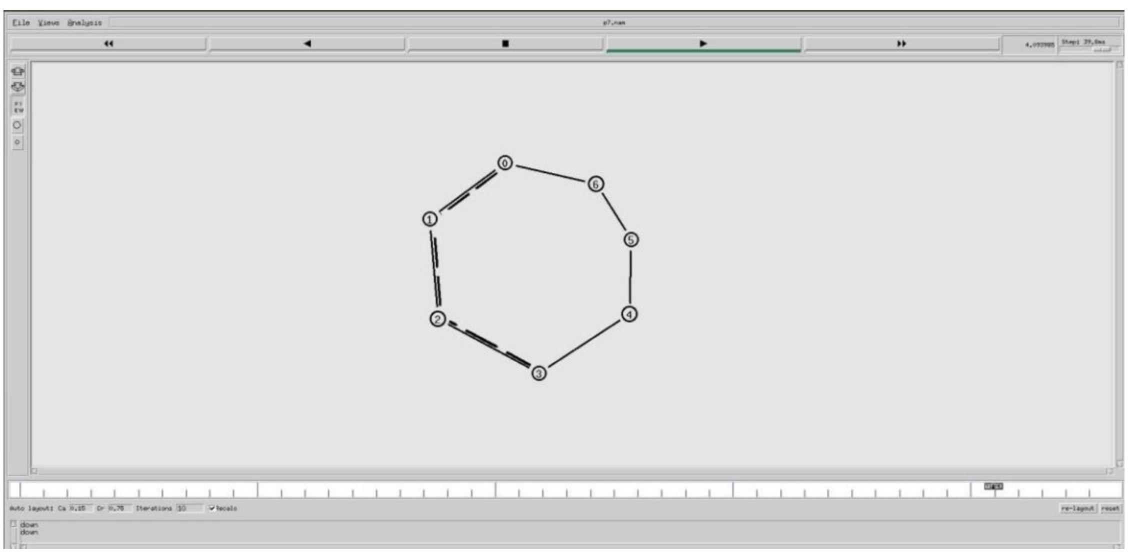
Aim: "Simulate a 6-node network to implement dynamic routing algorithm and verify its functionality".

TCL Code

```
#Create a simulator object
set ns [new Simulator]
#Tell the simulator to use dynamic routing
#Distance vector routing is an asynchronous algorithm in which node x sends the copy of its
distance vector to all its neighbors. When node x receives the new distance vector from one of
its #neighboring vector, v, it saves the distance vector of v and uses the Bellman-Ford equation
to update its own distance vector.
$ns rtproto DV
#Open the nam trace file
set nf [open p7.nam w]
$ns namtrace-all $nf
#Define a 'finish' procedure
proc finish {} {
global ns nf
$ns flush-trace
#Close the trace file
close $nf
#Execute nam on the trace file
exec nam p7.nam &
exit 0
}
#Create seven nodes
for {set i 0} {$i < 7} {incr i} {
set n($i) [$ns node]
}
#Create links between the nodes
for {set i 0} {$i < 7} {incr i} {
$ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}
#Create a UDP agent and attach it to node n(0)
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $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 Null agent (a traffic sink) and attach it to node n(3)
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0
#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
#Schedule events for the CBR agent and the network dynamics
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

NAM Output



Experiment 8

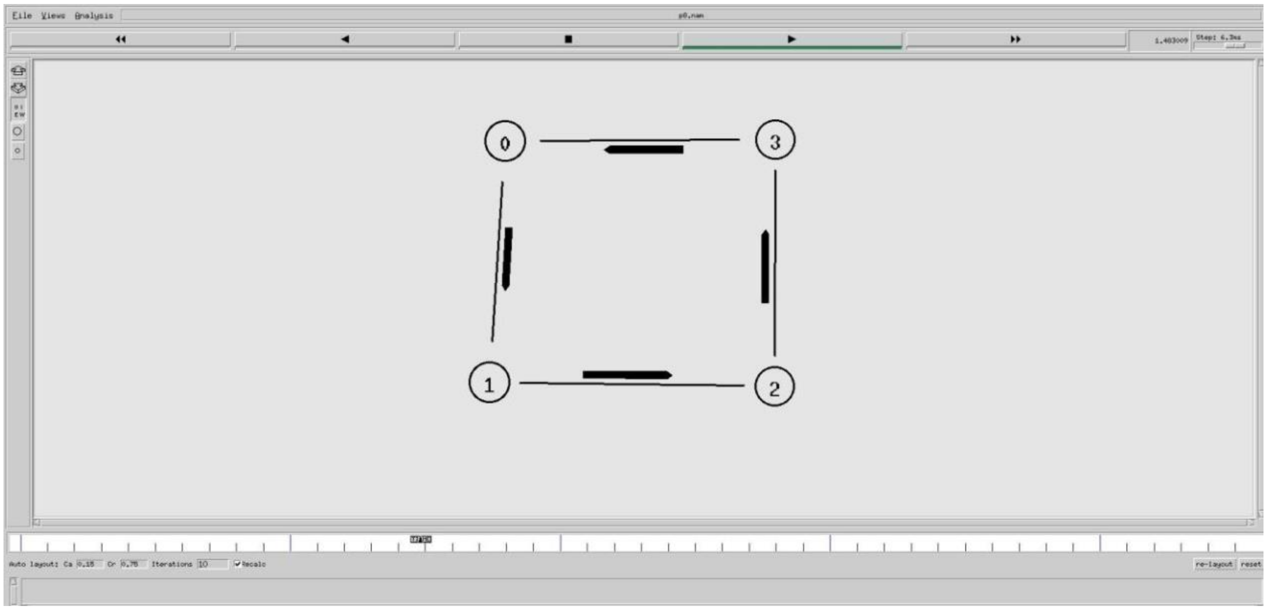
Aim: "Implement a method of cyclic data transmission using UDP protocol".

TCL code

```
set ns [new Simulator]
set nf [open p8.tr w]
$ns trace-all $nf
set ntrace [open p8.nam w]
$ns namtrace-all $ntrace
for {set i 0} { $i<4 } {incr i} {
set n($i) [$ns node] }
for {set i 0} { $i<4 } {incr i} {
$ns duplex-link $n($i) $n([expr ($i+1)%4]) 1Mb 10ms DropTail }
set udp [new Agent/UDP]
set null [new Agent/Null]
$ns attach-agent $n(0) $udp
$ns attach-agent $n(1) $null
$ns connect $udp $null
set cbr [new Application/Traffic/CBR]
$cbr set interval_ 0.005
$cbr set packetSize_ 500
$cbr attach-agent $udp
set udp1 [new Agent/UDP]
set null1 [new Agent/Null]
$ns attach-agent $n(1) $udp1
$ns attach-agent $n(2) $null1
$ns connect $udp1 $null1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set interval_ 0.005
$cbr1 set packetSize_ 500
$cbr1 attach-agent $udp1
set udp2 [new Agent/UDP]
set null2 [new Agent/Null]
$ns attach-agent $n(2) $udp2
$ns attach-agent $n(3) $null2
$ns connect $udp2 $null2
set cbr2 [new Application/Traffic/CBR]
$cbr2 set interval_ 0.005
```

```
$cbr2 set packetSize_ 500
$cbr2 attach-agent $udp2
set udp3 [new Agent/UDP]
set null3 [new Agent/Null]
$ns attach-agent $n(3) $udp3
$ns attach-agent $n(0) $null3
$ns connect $udp3 $null3
set cbr3 [new Application/Traffic/CBR]
$cbr3 set interval_ 0.005
$cbr3 set packetSize_ 500
$cbr3 attach-agent $udp3
proc Finish { } {
global ns nf ntrace
$ns flush-trace
close  $nf
close $ntrace
exec nam p8.nam &
exit 0
}
$ns at 0.5 "$cbr start"
$ns at 4.5 "$cbr stop"
$ns at 0.5 "$cbr1 start"
$ns at 4.5 "$cbr1 stop"
$ns at 0.5 "$cbr2 start"
$ns at 4.5 "$cbr2 stop"
$ns at 0.5 "$cbr3 start"
$ns at 4.5 "$cbr3 stop"
$ns at 5.0 "Finish"
$ns run
```

NAM Output



Experiment 9

Aim: "Implement using C, the error detecting code CRC for 16 bits".

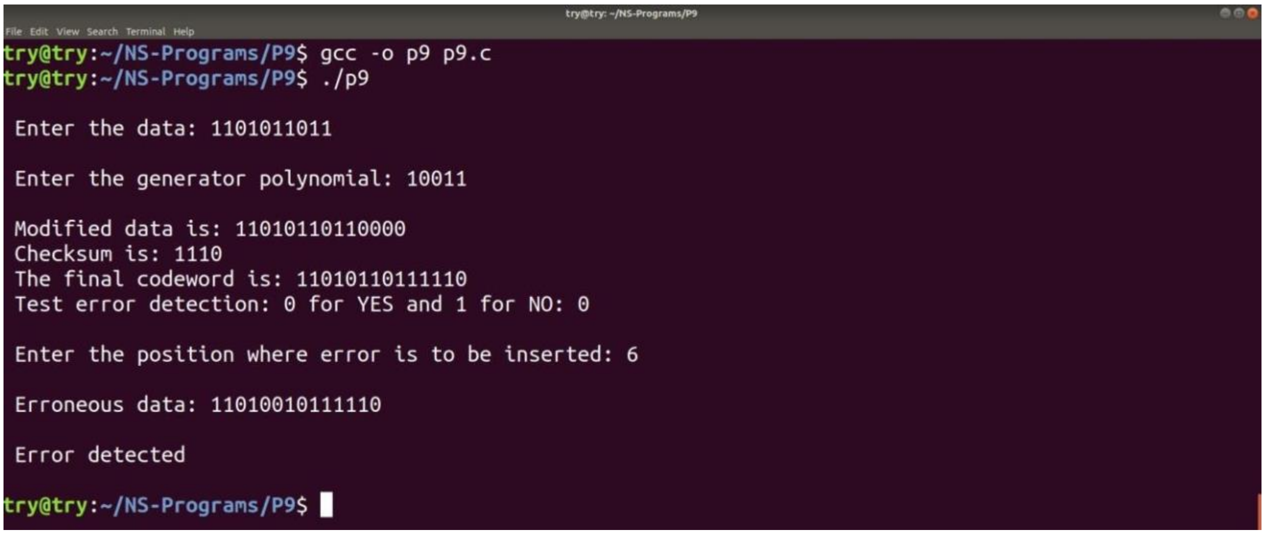
C Program for CRC for 5 bits

```
#include<stdio.h>
#include<string.h>
#define N strlen(g)
//declare the header libraries
char t[50], cs[50], g[50];
int a,e,c;
void xor()
{
    for(c=1;c<N;c++)
    //
            cs[c]=((cs[c]==g[c])?'0':'1');
            //Checking the XOR operation. If both operands are same, then output will b "0"
otherwise its "1".
}


void crc()
{
    for(e=0;e<N;e++)
    //Consider only first FIVE bits from the modified data
            cs[e]=t[e];
            //Copy those first FIVE bits to CHECKSUM cs[e] from t[e]
    do{
            if(cs[0]=='1')
            //If first leftmost bit is 1 then perform XOR operation
                    xor();
                    //Calling XOR function
            for(c=0;c<N-1;c++)
            //Performing XOR operation at the first iteration for FIVE bits (0 to N-1)
                    cs[c]=cs[c+1];
                    //Peform the same for all the data by right shift by 1
    cs[c]=t[e++];
    } while(e<=a+N-1);
    //Continue the operation for the entire data.
}

int main(){
    printf("\n Enter the data: ");
    //Enter the data as 1101011011
    scanf("%s", t);
```

```
// Data stored in a string t
printf("\n Enter the generator polynomial: ");
scanf("%s", g);
//Enter the generator polynomial: Since we have hard coded the GP as 10011
a=strlen(t);
// "a" defines the total length of the data
for(e=a;e<a+N-1;e++)
//Appending N-1 zeros to the data where N is the length of the GP
        t[e]='0';
        //t[e] defines appending zeros from e=a;e<a+N-1;e++
printf("\n Modified data is: %s", t);
//MOdified data is 11010110110000
crc();
//Call CRC function
printf("\n Checksum is: %s", cs);
//Print the checksum after XOR operation
for(e=a;e<a+N-1;e++)
//To append the checksum value instead of N-1 zeros in total length of the data
        t[e]=cs[e-a];
        //The remodified data with checksum (FINAL CODEWORD)
printf("\n The final codeword is: %s", t);
//Print the final codeword
printf("\n Test error detection: 0 for YES and 1 for NO: ");
//To check for error detection
scanf("%d", &e);
if(e==0)
//If the value of "e" is 0
{
do {
        printf("\n Enter the position where error is to be inserted: ");
        //Specify the position
        scanf("%d", &e);
        //Say for example, e=6
} while(e==0||e>a+N-1);
//WHILE states the boundary, means ranging for 0 to a+N-1

t[e-1] = (t[e-1]=='0')?'1':'0';
//Changing the bit from 0 to 1 and vice versa for error detection
printf("\n Erroneous data: %s\n",t);
}
crc();
for(e=0; (e<N-1)&&(cs[e]!='1'); e++);
//If CHECKSUM is not equal to 1 then error is detected else no error
        if(e<N-1)
                printf("\n Error detected \n \n");
```

```
        else
                printf("\n No error detected \n \n");
return 0;
}
```

Terminal Output



Note: The above code gives CRC output for 5bits generator polynomial. Students need to develop C code for 16bits generator polynomial CRC.

Experiment 10

Aim: "Implement using C, Hamming Code generation for error detection and correction"

C Program

```
#include<stdio.h>
int data[4],encoded[7],edata[7],syn[3];
int gmatrix[4][7]={{0,1,1,1,0,0,0},{1,0,1,0,1,0,0},{1,1,0,0,0,1,0},{1,1,1,0,0,0,1}};
int hmatrix[3][7]={{1,0,0,0,1,1,1},{0,1,0,1,0,1,1},{0,0,1,1,1,0,1}};
int main(){
int i,j;
printf("Hamming Code encoding\n");
printf("Enter the 4 bit data (one by one): \n");
for(i=0;i<4;i++)scanf("%d",&data[i]);
printf("Generator Matrix\n");
for(i=0;i<4;i++){
for(j=0;j<7;j++){
printf("%d",gmatrix[i][j]);}
printf("\n");
}
printf("\n\nEncoded data : ");
for(i=0;i<7;i++){
for(j=0;j<4;j++)encoded[i]^=(data[j]*gmatrix[j][i]);
printf("%d",encoded[i]);
}

printf("\n\nHamming Code Decoding  \n\n");
printf("Enter the encoded bit receieved (one by one) :\n");
for(i=0;i<7;i++)scanf("%d",&edata[i]);


printf("Syndrome = ");
for(i=0;i<3;i++){
for(j=0;j<7;j++)syn[i]^=(edata[j]*hmatrix[i][j]);
printf("%d",syn[i]);
}

for(j=0;j<=7;j++)
if(syn[0]==hmatrix[0][j]&&syn[1]==hmatrix[1][j]&&syn[2]==hmatrix[2][j])break;

if(j==7)printf("\n\nThe code is error free\n");
else{
printf("\n\nError Receieved at bit no %d of the data\n\n",j+1);
edata[j]=!edata[j];
```

```
printf("The correct data should be : ");
for(i=0;i<7;i++)printf("%d",edata[i]);
}
printf("\n\n");
return 0;
}
```

Terminal Output

```
try@try:~/NS-Programs/P10$ gcc -o p10 p10.c
try@try:~/NS-Programs/P10$ ./p10
Hamming Code encoding
Enter the 4 bit data (one by one):
1
0
0
1
Generator Matrix
0111000
1010100
1100010
1110001


Encoded data : 1001001

Hamming Code Decoding

Enter the encoded bit receieved (one by one) :
1
0
0
1
0
0
1
Syndrome = 000

The code is error free


try@try:~/NS-Programs/P10$
```

Experiment 11

AWrite a program to create mobile nodes using Destination-Sequenced Distance-Vector Routing (DSDV) protocol

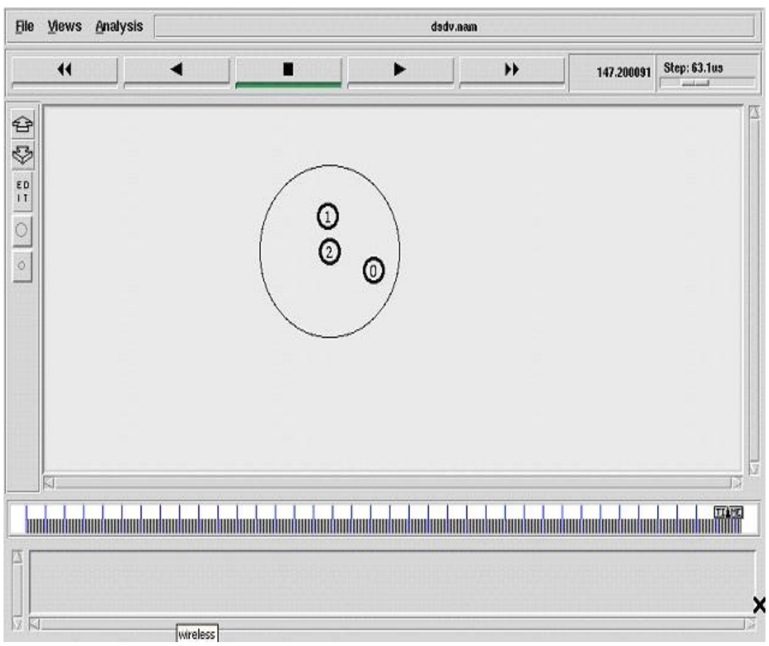
```
# Define setting option
set val(chan)     Channel/WirelessChannel   ;# channel type
set val(prop)     Propagation/TwoRayGround  ;# radio-propagation model
set val(netif)    Phy/WirelessPhy           ;# network interface type
set val(mac)      Mac/802_11                ;# MAC type
set val(ifq)      Queue/DropTail/PriQueue   ;# interface queue type
set val(ll)       LL                        ;# link layer type
set val(ant)      Antenna/OmniAntenna       ;# antenna model
set val(ifqlen)   50                        ;# max packet in ifq
set val(nn)       3                         ;# number of mobilenodes
set val(rp)       DSDV                      ;# routing protocol
set val(x)        500                       ;# X dimension of topography
set val(y)        400                       ;# Y dimension of topography
set val(stop)     150                       ;# time of simulation end
```

```
#Creating trace file and nam file
set tracefd       [open dsdv.tr w]
set windowVsTime2 [open win.tr w]
set namtrace      [open dsdv.nam w]

$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# set up topography object
set topo       [new Topography]

$topo load_flatgrid $val(x) $val(y)

create-god $val(nn)

# configure the nodes
        $ns node-config -adhocRouting $val(rp) \
                   -llType $val(ll) \
                   -macType $val(mac) \
                   -ifqType $val(ifq) \
                   -ifqLen $val(ifqlen) \
                   -antType $val(ant) \
                   -propType $val(prop) \
                   -phyType $val(netif) \
                   -channelType $val(chan) \
                   -topoInstance $topo \
```

```
-
agentTrace ON
\
-
routerTrace
ON \
-
macTrace OFF
\
-
movementTrac
e ON

for {set i
0} {$i <
$val(nn) } {
incr i } {
set
node_($i) [$ns
node]
}

# Provide
initial location
of
mobilenodes
$node_(0) set
X_ 5.0
$node_(0) set
Y_ 5.0
$node_(0) set
Z_ 0.0

$node_(1) set
X_ 490.0
$node_(1) set
Y_ 285.0
$node_(1) set
Z_ 0.0

$node_(2) set
X_ 150.0
$node_(2) set
Y_ 240.0
$node_(2) set
Z_ 0.0
```

```
# Generation of movements
$ns at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns at 110.0 "$node_(0) setdest 480.0 300.0 5.0"

# Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns attach-agent $node_(0) $tcp
$ns attach-agent $node_(1) $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 10.0 "$ftp start"

# Printing the window size
proc plotWindow {tcpSource file} {
global ns
set time 0.01
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 10.1 "plotWindow $tcp $windowVsTime2"

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 30 defines the node size for nam
$ns initial_node_pos $node_($i) 30
}

# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
    $ns at $val(stop) "$node_($i) reset";
}

# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "stop"
$ns at 150.01 "puts \"end simulation\" ; $ns halt"
proc stop {} {
    global ns tracefd namtrace
    $ns flush-trace
    close $tracefd
    close $namtrace
```

```
exec nam
dsdv.nam &
exit 0
}

$ns run

# How to run
the program:

$ns dsdv.tcls
```



Experiment 12

Aim: "Simulate a 7-node network to verify Link State routing protocol".

```
set ns [new Simulator]
set namfile [open p12.nam w]
$ns namtrace-all $namfile
set tracefile [open p12.tr w]
$ns trace-all $tracefile
proc finish {} {
global ns namfile tracefile
$ns  flush-trace
close  $namfile
close $tracefile
exec nam p12.nam &
exit 0
}
set n0 [$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

#The Link state routing algorithm is also known as Dijkstra's algorithm which is used to find the shortest path from one node to every other node in the network.

$ns rtproto LS
$ns rtmodel-at 20.0 down $n1 $n4
$ns rtmodel-at 23.0 up $n1 $n4
$ns rtmodel-at 25.0 down $n2 $n4
$ns rtmodel-at 40.0 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"
$ns at 2.0 "$cbr1 start"
$ns at 45 "finish"
$ns run
```

NAM Output