# WIRELESS NETWORKS

# LAB MANUAL

(code=22CY805)

**Batch - 18**

| **Roll Number** | **Name/signature** |
| --- | --- |
| 231FA19011 | MUBEENA/ |
| 231FA19032 | SUSMITA/ |
| 231FA19038 | SANTHOSH/ |
| 231FA19042 | HANEESH/ |

Faculty Signature

## Table of Contents:

1.Wireless TCP Experiment using NSG2 in Ubuntu[15/7/25]  
1.1 Introduction to TCP in NSG2  
1.2 Simulation Setup  
1.3 Network Topology and Parameters  
1.4 Execution and Results  
1.5 Analysis of TCP Performance

2Wireless UDP Experiment using NSG2 in Ubuntu[22/7/25]  
2.1 Introduction to UDP in NSG2  
2.2 Simulation Setup  
2.3 Network Topology and Parameters  
2.4 Execution and Results  
2.5 Analysis of UDP Performance

3.Mobile node[5/8/25]  
3.1 Concept of Node-to-Node Transmission  
3.2 Configuring Two Nodes in NSG2  
3.3 Specifying Distance between Nodes  
3.4 Running the Simulation  
3.5 Observations and Results

4.Experiment 4 – Link State Routing (LSR) Wired in NS2 (via NSG2)[12/8/25]  
4.1 Objective  
4.2 Topology & Parameters  
4.3 NS2/NSG2 Steps  
4.4 TCL Snippet (core)  
4.5 Expected Results & Analysis

5.Experiment 5 – Distance Vector Routing (DVR) Wired in NS2 (via NSG2)[19/8/25]

5.1 Objective  
5.2 Topology & Parameters  
5.3 TCL Snippet (core)  
5.4 Expected Results & Analysis

# **Experiment –1**

# **TCP Communication using NS2**

### **Aim:**

To simulate **TCP communication** between nodes in NS2 and analyze performance metrics such as throughput, delay, and packet delivery.

### **Objective:**

1.To understand how TCP works in a simulated environment.

2.To learn the use of NS2 and NSG2 for designing network topologies.

3.To analyze the behavior of TCP under different network conditions.

4.To visualize TCP packet flow using NAM (Network Animator).

### **Theory / Background:**

**1.NS2 (Network Simulator 2)** is an open-source simulation tool widely used for networking research.

**2.TCP (Transmission Control Protocol)** is a connection-oriented protocol that ensures reliable data transfer with error-checking and acknowledgment mechanisms.

3.In NS2, TCP connections can be simulated between nodes by attaching TCP agents and traffic generators.

4.Performance metrics like throughput, delay, and packet loss can be analyzed using trace files.

### **Requirements:**

**1.Hardware**: Any modern PC/laptop

**2.Software**:

i)Ubuntu (20.04 or later recommended)

ii)NS2 installed (sudo apt-get install ns2)

iii)NSG2 (Network Simulation Graphical Tool for NS2)

iv)NAM (Network Animator)

### **Procedure / Steps:**

1.Install **NS2** and **NSG2** in Ubuntu.

2.Open NSG2 and create a topology with:

i)At least **4 nodes**.

ii)One TCP source node

iii)One TCP sink (receiver) node.

iv)Two intermediate router nodes (optional).

3.Attach a **TCP agent** to the source and a **sink agent** to the destination.

4.Connect nodes with links (e.g., 1Mb, 10ms).

5.Generate a **traffic flow** using FTP over TCP.

6.Save the file as tcp.tcl.

7.Run the simulation with:

ns tcp.tcl

8.Open the **NAM output file** to visualize packet flow:

nam tcp.nam

9.Analyze results from the **trace file (**tcp.tr**)** for throughput, delay, and packet loss.

10.Plot graphs (using AWK scripts, Xgraph, or Python/Excel).

**Code:**

# TCP Simulation in NS2

# Create simulator

set ns [new Simulator]

# Open trace files

set tf [open tcp.tr w]

set nf [open tcp.nam w]

$ns trace-all $tf

$ns namtrace-all $nf

# Create nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

# Create links

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

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

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

# Setup TCP connection

set tcp [new Agent/TCP]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n3 $sink

$ns connect $tcp $sink

# Setup FTP over TCP

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ns at 1.0 "$ftp start"

$ns at 4.0 "$ftp stop"

# Finish procedure

proc finish {} {

global ns nf tf

$ns flush-trace

close $tf

close $nf

exec nam tcp.nam &

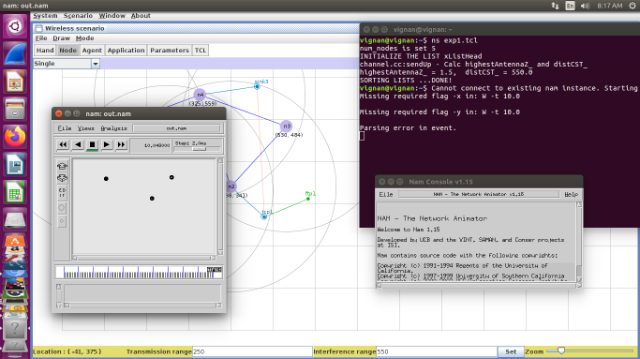
exit 0

}

$ns at 5.0 "finish"

$ns run

OUTPUT:



### **Results:**

1.NAM animation shows TCP packets flowing between nodes.

2.Trace file records events like send, receive, drop.

3.Graphs can be plotted for:

**4.Throughput** (packets/sec or bits/sec).

**5.End-to-End Delay**.

**6.Packet Delivery Ratio**.

### **Conclusion:**

1.TCP communication between nodes was successfully simulated using NS2.

2.The simulation showed **reliable transmission with acknowledgments**, as expected from TCP.

3.Performance metrics such as throughput and delay were analyzed from the trace file.

4.This experiment helped understand how TCP behaves in a controlled simulation environment.

# **Experiment-2**

# **UDP Communication using NS2**

### **Aim:**

To simulate **UDP communication** between nodes in NS2 and analyze performance metrics such as throughput, delay, and packet delivery.

### **Objective**

1.To understand how UDP works in a simulated environment.

2.To learn the use of NS2 and NSG2 for designing network topologies.

3.To compare the behavior of UDP with TCP in terms of reliability and performance.

4.To visualize UDP packet flow using NAM (Network Animator).

### **Theory / Background:**

1. **UDP (User Datagram Protocol)** is a **connectionless protocol** that provides faster communication but does not guarantee reliability (no acknowledgments, retransmissions, or congestion control).

2.Unlike TCP, UDP is lightweight and suitable for applications where speed is critical (e.g., video streaming, online gaming, VoIP).

3. In NS2, UDP agents and Constant Bit Rate (CBR) traffic generators are used to simulate UDP communication.

4. Since UDP does not ensure delivery, packet drops may occur under heavy load or network congestion.

### **Requirements:**

**Hardware**: Any modern PC/laptop

**Software**:Ubuntu (20.04 or later recommended)

NS2 installed (sudo apt-get install ns2)

NSG2 (Network Simulation Graphical Tool for NS2)

NAM (Network Animator)

### **Procedure / Steps:**

1. Install **NS2** and **NSG2** in Ubuntu.
2. Open NSG2 and create a topology with:
3. At least **4 nodes**.
4. One UDP source node.
5. One UDP sink (receiver) node.
6. Two intermediate router nodes (optional).
7. Attach a **UDP agent** to the source and a **Null agent** to the destination.
8. Connect nodes with links (e.g., 1Mb, 10ms).
9. Generate a **traffic flow** using CBR over UDP.
10. Save the file as udp.tcl.
11. Run the simulation with:

ns udp.tcl

12.Open the **NAM output file** to visualize packet flow:

nam udp.nam

13.Analyze results from the **trace file (**udp.tr**)** for throughput, delay, and packet loss.  
 14.Compare results with TCP simulation.

**CODE:**

# UDP Simulation in NS2

# Create simulator

set ns [new Simulator]

# Open trace files

set tf [open udp.tr w]

set nf [open udp.nam w]

$ns trace-all $tf

$ns namtrace-all $nf

# Create nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

# Create links

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

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

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

# Setup UDP connection

set udp [new Agent/UDP]

$ns attach-agent $n0 $udp

set null [new Agent/Null]

$ns attach-agent $n3 $null

$ns connect $udp $null

# Setup CBR over UDP

set cbr [new Application/Traffic/CBR]

$cbr set packetSize\_ 500

$cbr set interval\_ 0.005

$cbr attach-agent $udp

$ns at 1.0 "$cbr start"

$ns at 4.0 "$cbr stop"

# Finish procedure

proc finish {} {

global ns nf tf

$ns flush-trace

close $tf

close $nf

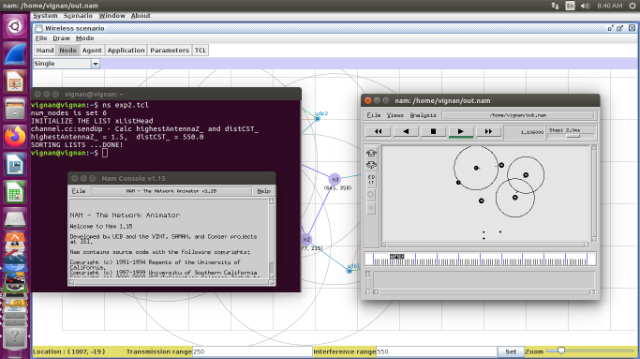
exec nam udp.nam &

exit 0

}$ns at 5.0 "finish"

$ns run

**OUTPUT:**

****

### **Results:**

NAM animation shows **UDP packets flowing** between nodes.

Trace file records events like send, receive, drop.

Graphs can be plotted for:

**Throughput** (packets/sec or bits/sec).

**End-to-End Delay**.

**Packet Loss Ratio**.

Unlike TCP, some packets may be dropped due to lack of retransmission.

### **Conclusion:**

### 1.UDP communication between nodes was successfully simulated using NS2.

### 2.The simulation showed that UDP provides **fast transmission** but **does not guarantee reliability**.

### 3.Packet drops were observed under certain conditions, which highlights the difference from TCP.

### 4.This experiment provided insights into why UDP is preferred for real-time applications like video/audio streaming.

# **Experiment-3**

# **Transmission between Two Nodes based on Distance in NS2**

### **Aim:**

To simulate **data transmission between two nodes** in NS2 while considering the distance between them, and to observe how distance affects performance parameters like throughput, delay, and packet delivery.

### **Objective:**

To configure a simple two-node network in NS2.

To study the effect of distance on transmission.

To understand how propagation delay changes with distance.

To measure performance metrics (delay, throughput, packet loss) as the distance between nodes varies.

### **Theory / Background:**

### In NS2, links between nodes can be configured with specific **bandwidth** (e.g., 1Mb) and **propagation delay** (e.g., 10ms).

### **Propagation delay** depends on the physical distance between nodes and the medium of transmission.

### Delay = Distance / Propagation Speed

### By changing the delay parameter in duplex-link commands, we simulate the effect of different distances between nodes.

### This helps us analyze how **larger distances introduce higher delays**, affecting overall communication.

### **Requirements:**

**Hardware**: Any modern PC/laptop

**Software**:Ubuntu (20.04 or later recommended)

NS2 installed (sudo apt-get install ns2)

NSG2 (Network Simulation Graphical Tool for NS2)

NAM (Network Animator)

### **Procedure / Steps:**

### 1.Install **NS2** and **NSG2** in Ubuntu.

### 2.Create a simple topology with **two nodes (n0, n1)**.

### 3.Connect them using a **duplex link** with specified bandwidth and delay

### 4.Attach a **TCP/UDP agent** to one node and a corresponding sink to the other node.

### 5.Generate traffic (FTP for TCP or CBR for UDP).

### 6.Vary the **delay parameter** in the duplex-link (to represent different distances).

### 7.Save the script as two\_nodes.tcl.

### 8.Run the simulation and view results in NAM.

### 9.Analyze the trace file for throughput, delay, and packet loss.

### 10.Compare results for short vs. long distances.

**CODE:**

# Two Node Transmission Simulation in NS2

# Create simulator

set ns [new Simulator]

# Open trace files

set tf [open two\_nodes.tr w]

set nf [open two\_nodes.nam w]

$ns trace-all $tf

$ns namtrace-all $nf

# Create nodes

set n0 [$ns node]

set n1 [$ns node]

# Create duplex link between two nodes

# Bandwidth = 1Mb, Delay = 10ms (representing distance)

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

# Setup UDP connection (can also use TCP)

set udp [new Agent/UDP]

$ns attach-agent $n0 $udp

set null [new Agent/Null]

$ns attach-agent $n1 $null

$ns connect $udp $null

# Setup CBR traffic over UDP

set cbr [new Application/Traffic/CBR]

$cbr set packetSize\_ 500

$cbr set interval\_ 0.005

$cbr attach-agent $udp

$ns at 1.0 "$cbr start"

$ns at 4.0 "$cbr stop"

# Finish procedure

proc finish {} {

global ns nf tf

$ns flush-trace

close $tf

close $nf

exec nam two\_nodes.nam &

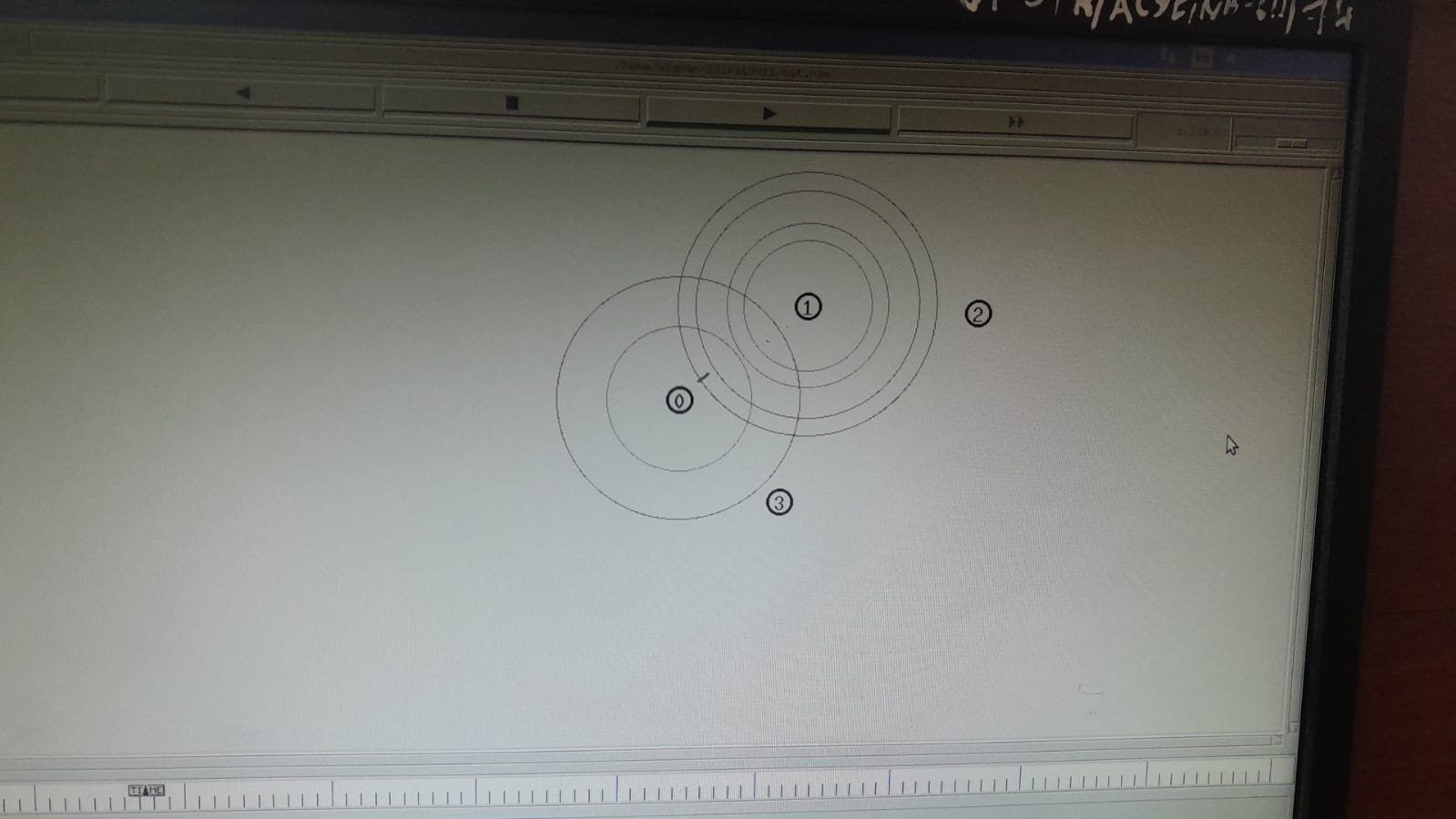
exit 0

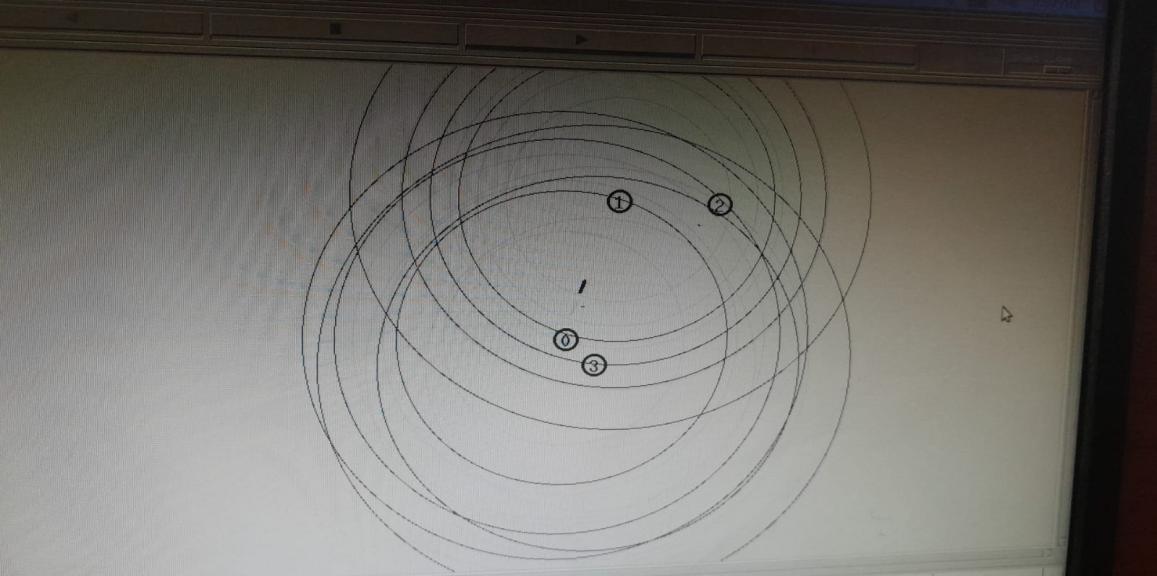
}

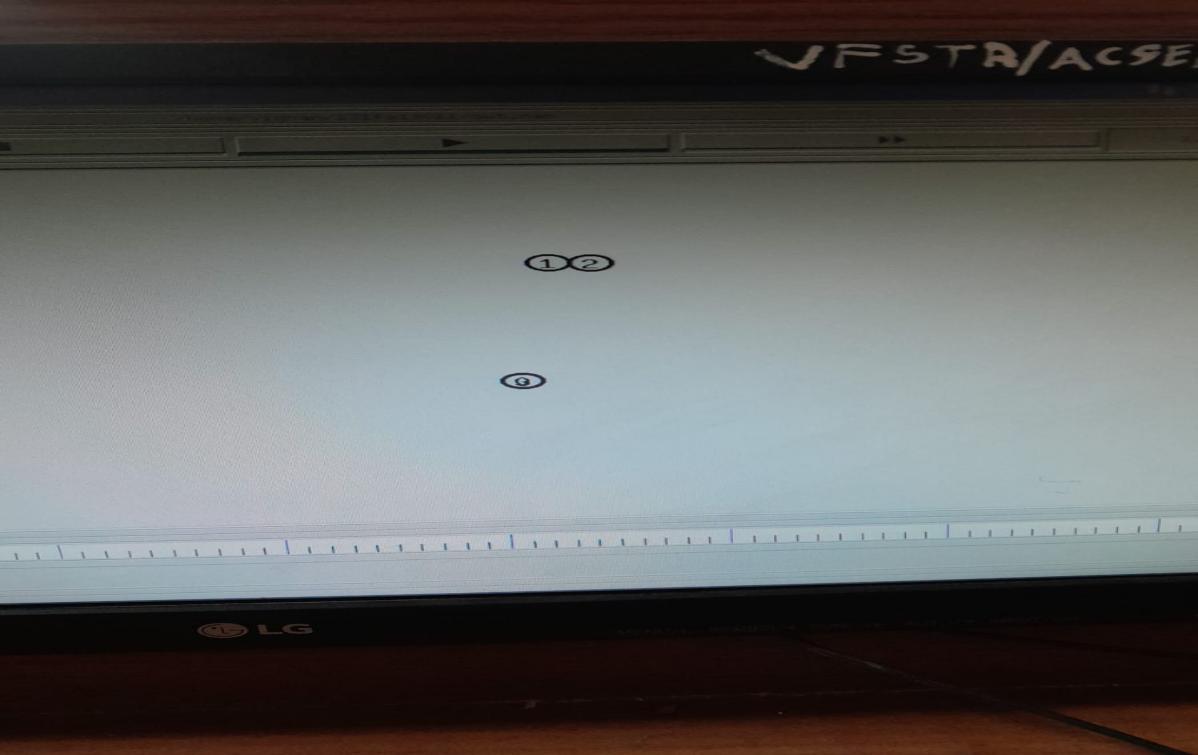
$ns at 5.0 "finish"

$ns run

**Output:**

****

****

****

### **Results:**

NAM shows direct transmission between two nodes.

With small delay (short distance), packets reach quickly.

With larger delay (long distance), end-to-end delay increases.

Graphs can be plotted for:

End-to-End Delay vs Distance

Throughput vs Distance

### **Conclusion:**

### Transmission between two nodes was successfully simulated.

### As distance (propagation delay) increases, **end-to-end delay increases** and throughput may reduce slightly.

### No packet loss occurs unless buffer overflow or congestion is introduced.

### This experiment demonstrates the impact of distance on network performance.

**IV. EXPERIMENT 4 – LINK STATE ROUTING IN NS2(Wired)**

# **A. Abstract**

This experiment simulates Link State Routing (LSR) and studies how it adapts to link failures.

# **B. Introduction**

LSR is a proactive routing protocol where routers maintain a global topology and compute shortest paths using Dijkstra’s algorithm.

# **C. Methodology**

* NS2 simulation initialized with Link State routing.
* Multiple interconnected nodes.
* A link was intentionally brought down and restored.

# **D. Results and Discussion**

Simulation showed immediate rerouting after link failure. Delay temporarily increased during the failure, but throughput recovered quickly due to fast convergence.

**Code:**

# Link State Routing Simulation in NS2

set ns [new Simulator

# Trace files

set tf [open lsr.tr w]

set nf [open lsr.nam w]

$ns trace-all $tf

$ns namtrace-all $nf

# Define nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

# Create links (all nodes connected in a mesh for LSR demonstration)

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

$ns duplex-link $n0 $n2 1Mb 20ms DropTail

$ns duplex-link $n1 $n3 1Mb 15ms DropTail

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

$ns duplex-link $n3 $n4 1Mb 5ms DropTail

# Setup UDP connection

set udp [new Agent/UDP]

$ns attach-agent $n0 $udp

set null [new Agent/Null]

$ns attach-agent $n4 $null

$ns connect $udp $null

# CBR traffic

set cbr [new Application/Traffic/CBR]

$cbr set packetSize\_ 500

$cbr set interval\_ 0.01

$cbr attach-agent $udp

$ns at 1.0 "$cbr start"

$ns at 4.0 "$cbr stop"

# Finish procedure

proc finish {} {

global ns nf tf

$ns flush-trace

close $tf

close $nf

exec nam lsr.nam &

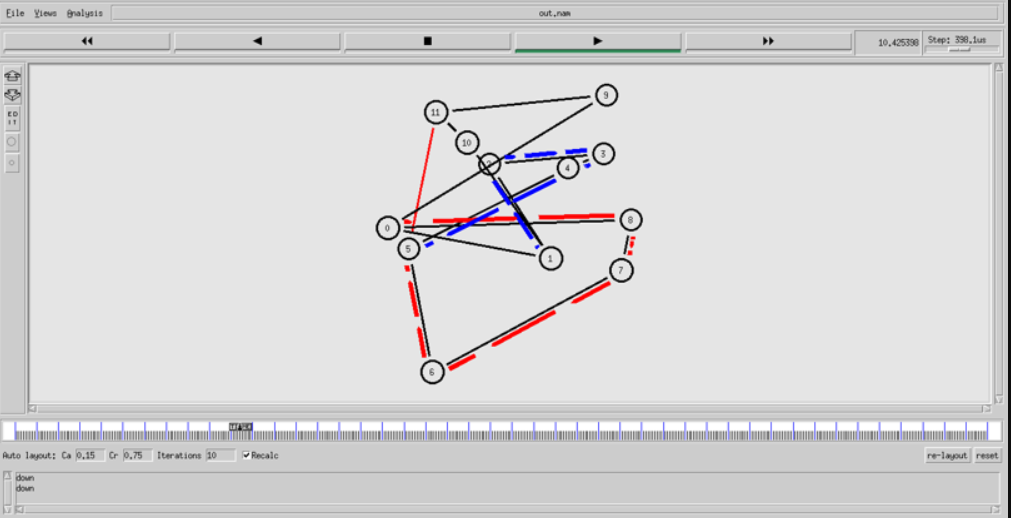
exit 0

}

$ns at 5.0 "finish"

$ns run

**Output:**



# **E. Conclusion**

LSR provided robust and adaptive routing with rapid recovery from failures.

**V. EXPERIMENT 5 – DISTANCE VECTOR ROUTING IN NS2(Wired)**

# **A. Abstract**

This experiment simulates Distance Vector Routing (DVR) and evaluates convergence after link failures.

# **B. Introduction**

DVR uses the Bellman-Ford algorithm and exchanges routing information with neighbors. It is simpler than LSR but slower in convergence.

# **C. Methodology**

* NS2 initialized with Distance Vector routing.
* Multiple nodes interconnected.
* Link failures were introduced to observe convergence.

**Code:**

# Distance Vector Routing Simulation in NS2

set ns [new Simulator]

# Trace files

set tf [open dvr.tr w]

set nf [open dvr.nam w]

$ns trace-all $tf

$ns namtrace-all $nf

# Define nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

# Create links

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

$ns duplex-link $n1 $n2 1Mb 15ms DropTail

$ns duplex-link $n2 $n3 1Mb 20ms DropTail

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

$ns duplex-link $n0 $n4 1Mb 50ms DropTail

# Setup TCP connection

set tcp [new Agent/TCP]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n3 $sink

$ns connect $tcp $sink

# FTP over TCP

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ns at 1.0 "$ftp start"

$ns at 4.0 "$ftp stop"

# Finish procedure

proc finish {} {

global ns nf tf

$ns flush-trace

close $tf

close $nf

exec nam dvr.nam &

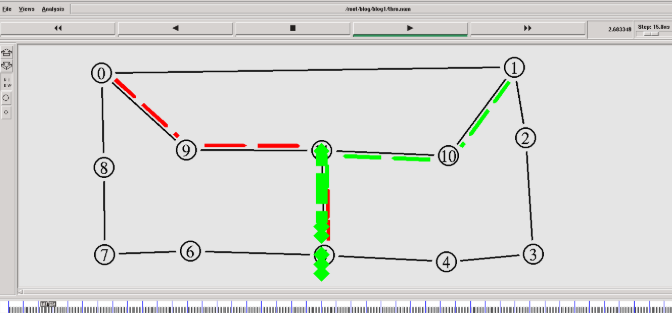
exit 0

}

$ns at 5.0 "finish"

$ns run

**Output:**

****

# **D. Results and Discussion**

DVR recomputed routes successfully after link failures but with noticeable convergence delays compared to LSR. Throughput dips and temporary packet loss were observed.

# **E. Conclusion**

DVR achieved successful routing but was less efficient than LSR in recovery time and performance stability.

1. **CONCLUSION**

This lab manual demonstrated the use of NS2 and NSG2 for simulating communication protocols and routing algorithms. TCP and UDP differ in reliability and speed, while Link State and Distance Vector routing differ in convergence and adaptability. The experiments provided hands-on understanding of network protocol behavior, equipping students with foundational knowledge for research and development in wireless networking.