

LAB MANUAL

DATA COMMUNICATIONS AND NETWORKS LAB



Prepared by

Mr. K. Srinivasa Rao

Assistant Professor Adhoc, ECE Department, JNTUA CEA

P. Md. Fayaz

III B.Tech ECE Student, Roll. No:22001A0421

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

JNTUA COLLEGE OF ENGINEERING (AUTONOMOUS)::ANANTAPUR

INDEX

S.No	EXPERIMENT	Page No
1	TELNET And FTP Between N Sources And N Sinks	3-16
2	EFFECT OF VARIOUS QUEUEING DISCIPLINES (RED /WEIGHTED RED / ADAPTIVE RED) ON NETWORK PERFORMANCE	17-31
3	SIMULATION OF FTP, HTTP, DBMS ACCESS IN NETWORKS	32-39
4	SIMULATION OF SLIDING WINDOW PROTOCOL	40-44
5	SIMULATION OF GO BACK N, SELECTIVE REPEAT PROTOCOLS	45-50
6	COMPARISION OF CSMA/CA AND CSMA/CD PROTOCOLS	51-60
7	IMPLEMENTATION OF A ROUTING ALGORITHM	61-65
8	SIMULATION OF CONGESTION CONTROL ALGORITHMS	66-75
9	SIMULATION OF DIFFERENT LAN TOPOLOGIES	76-90
10	INTERCONNECTING LANS USING SWITCH, HUB, ROUTER	91-106

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Vision:

To produce globally competitive and socially sensitized engineering graduates and to bring out quality research in the frontier areas of Electronics & Communication Engineering.

Mission:

M1: To provide quality and contemporary education in the domain of Electronics & Communication Engineering through a periodically updated curriculum, state-of-the-art laboratory facilities, collaborative ventures with industries, and an effective teaching-learning process, while fostering professional ethics, social responsibility, and a commitment to sustainable development.

M2: To impart essential knowledge, foster research, and promote innovative technologies in Electronics & Communication Engineering, while cultivating teamwork, and ethical practices in students to meet the evolving expectations of industry and society

Program Educational Objectives:

PEO1: Graduates will apply core principles of electronics and communication engineering to analyse, design, and develop innovative solutions for complex engineering problems

PEO2: Graduates will adapt to emerging technologies such as IoT, VLSI, AI/ML, 5G /6G continuously upgrading their knowledge through research, higher education and professional development.

PEO3: Graduates will integrate electronics, communication, and computing technologies to contribute to interdisciplinary projects, fostering entrepreneurship and innovation for societal and industrial advancements.

PEO4: Graduates will demonstrate effective communication, teamwork, leadership and ethical responsibility contributing to sustainable development and addressing global challenges through engineering solutions.

Program Outcomes:

- PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and engineering specialization to the solution of complex engineering problems.
- PO2. Problem analysis:** Identify, formulate, research literature, and analyze engineering problems to arrive at substantiated conclusions using first principles of mathematics, natural, and engineering sciences.
- PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components, processes to meet the specifications with consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- PO4. Conduct investigations of complex problems:** Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- PO9. Individual and team work:** Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
- PO10. Communication:** Communicate effectively with the engineering community and with society at large. Be able to comprehend and write effective reports documentation. Make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes:

PSO1: Ability to design, implement, and analyze complex electronics and communication systems, signal processing techniques, and embedded/VLSI circuits for real-world applications.

PSO2: Ability to apply modern tools and emerging technologies like IoT, Artificial Intelligence (AI), Machine Learning (ML), and 5G/6G in the design and development of cutting-edge electronics and communication systems.

PSO3: Ability to work in interdisciplinary projects that integrate electronics, communication, computer science and other branches of Engineering to create innovation solutions.

Course Outcomes:

At the end of the course, the student will be able to

COs	STATEMENT	BTL
CO1	Design and Implement Telnet and FTP Traffic Simulation Across Multiple Sources and Sinks.	3
CO2	Analyze the Effect of Queueing Disciplines (RED, Weighted RED, Adaptive RED) on Network Congestion.	4
CO3	Simulate and Evaluate the Impact of VLAN Configuration on Network Performance	4
CO4	Synthesize the Behavior of CSMA/CA and CSMA/CD Protocols in Wired and Wireless Networks	5
CO5	Design and Implement a Routing Algorithm for Dynamic Network Conditions	5

Course Articulation Matrix

COs	POs												PSOs		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1			3		3	2							3	3	2
CO2		3		3	2								3	2	2
CO3			3	3		2							3	2	3
CO4	3				3		2						3	3	3
CO5			3	3		2							3	3	3

List of Experiments:

- Introduction to Computer Network laboratory
- Introduction to Discrete Event Simulation
- Discrete Event Simulation Tools -ns2/ns3, Omnet++

Usage of the tool ns2/ns3to:

1. Simulate telnet and ftp between N sources
- N sinks (N = 1, 2, 3). Evaluate the effect of increasing data rate on congestion.
2. Simulating the effect of queueing disciplines on network performance-
Random Early Detection/Weighted RED / Adaptive RED (This can be used as a lead up to DiffServ / IntServlater).

3. Simulate http, ftp and DBMS access in networks
4. Effect of VLAN on network performance –i) multiple VLANs and single router ii) multiple VLANs with separate multiple routers
5. Implementation of IP address configuration.
6. To create scenario and study the performance of network with CSMA/CA protocol and compare with CSMA/CD protocols.
7. Implementation of a routing algorithm
8. Simulation of Congestion Control Algorithms
9. Simulating the effect of DiffServ/ IntServ in routers on throughput enhancement.
10. Simulating the performance of wireless networks
11. Case Study I: Evaluating the effect of Network Components on Network Performance to Design and Implement LAN With Various Topologies and To Evaluate Network Performance Parameters for DBMS etc.)
12. Case Study II: Evaluating the effect of Network Components on Network Performance to Design and Implement LAN Using Switch/Hub/Router as Interconnecting Devices for Two Different LANs and To Evaluate Network Performance Parameters.

Note: At least 10 Experiments out of the list must be done in the semester.

References:

Online Learning Resources/Virtual Labs:

LAB INSTRUCTIONS

1. General Instructions

- Students must report to the lab **on time** and **sign the attendance register**.
- Maintain **discipline and silence** throughout the session.
- Bring your **observation book, lab record**, and **pen/pencil box** to every lab class.
- **Mobile phones** must be kept away unless explicitly permitted for experiment purposes.

2. Preparation Before Lab

- Read and understand the **theory behind the experiment** from the manual.
- Complete the **pre-lab assignment or program logic** as instructed.
- Review related **protocols, OSI layers, topologies**, and basic concepts.

3. During the Lab

- **Strictly follow the experiment instructions** given by the faculty/lab assistant.
- Perform only the **assigned experiments** using the appropriate tools and software.
- **Handle network components (like routers, switches, cables) with care.**
- Use simulation tools (like **Cisco Packet Tracer / NS2 / Wireshark / NetSim**) only as instructed.

4. Observation & Record

- All output screenshots, waveforms, and tabulated results must be recorded in the **observation book** during the lab.
- The **lab record** must be updated regularly and signed by the faculty every week.
- Use proper format:
 - **Aim**
 - **Equipment/Software Used**
 - **Theory**
 - **Algorithm/Procedure**
 - **Program/Simulation Screenshots**
 - **Output**
 - **Result**
 - **Viva Questions**

5. Safety and Ethical Usage

- Do not tamper with lab network settings, IP configurations, or cables.
- Avoid using the lab internet for **non-academic purposes**.
- Save your work appropriately and **log out** after use.

6. Post Lab

- **Submit the completed record** within the deadline.
- Attend **viva-voce**, which will assess your understanding of:
 - Protocols (TCP/IP, UDP, FTP, etc.)
 - Network Devices
 - Routing Algorithms
 - Addressing (IP, MAC)
 - Layer-wise Communication

EXPERIMENT 1**TELNET AND FTP
BETWEEN N SOURCES AND N SINKS**

AIM: To Simulate a Program of TELNET and FTP between N Sources – N Sinks

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:

Telnet and FTP (File Transfer Protocol) are two protocols that facilitate remote communication and file transfer over a network.

1. Telnet (Telecommunication Network)

Telnet is a protocol used to establish a remote command-line interface to a host over a network. It enables users to log in to another computer and execute commands as if they were physically present at that machine.

- **Functionality:**
 - Allows remote terminal access.
 - Uses TCP (Transmission Control Protocol) on port **23**.
 - Does not encrypt data, making it insecure for sensitive information.
 - Often replaced by SSH (Secure Shell) for secure remote access.

2. FTP (File Transfer Protocol)

FTP is a standard protocol used to transfer files between computers over a network.

- **Functionality:**
 - Uses TCP ports **20 (data transfer)** and **21 (control connection)**.
 - Can operate in **active** or **passive** mode.
 - Supports user authentication but is generally insecure unless secured by FTPS (FTP Secure) or SFTP (SSH File Transfer Protocol).

PROGRAM:

```
#=====

#Simulation parameters setup #=====

set val(stop)      10.5  ;# time of simulation end
#=====

#Initialization #=====

#Create a ns simulator
set ns [new Simulator]

#Open the NS trace file
set tracefile [open telnet.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open telnet.nam w]
$ns namtrace-all $namfile

#=====

#Nodes Definition #=====

#Create 6 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

#=====
```

```

#      Links Definition #=====
#Create links between nodes
$ns duplex-link $n0 $n2 100.0Mb 10ms RED
$ns queue-limit $n0 $n2 50
$ns duplex-link $n3 $n2 100.0Mb 10ms RED
$ns queue-limit $n3 $n2 50
$ns duplex-link $n1 $n2 100.0Mb 10ms RED
$ns queue-limit $n1 $n2 50
$ns duplex-link $n3 $n4 100.0Mb 10ms RED
$ns queue-limit $n3 $n4 50
$ns duplex-link $n3 $n5 100.0Mb 10ms RED
$ns queue-limit $n3 $n5 50

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n3 $n2 orient left

$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down

#=====
#Agents Definition #=====
#Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink3 [new Agent/TCPSink]
$ns attach-agent $n5 $sink3
$ns connect $tcp0 $sink3
$tcp0 set packetSize_ 1500 #Setup a TCP connection

```

```
set tcp1 [new Agent/TCP]
$ns attach-agent $n4 $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n1 $sink2
$ns connect $tcp1 $sink2
$tcp1 set packetSize_ 1500

#=====

#Applications Definition #=====

#Setup a TELNET Application over TCP connection
set telnet0 [new Application/Telnet]
$telnet0 attach-agent $tcp0
$ns at 1.0 "$telnet0 start"
$ns at 10.0 "$telnet0 stop"
$telnet0 set interval_ 0.002
$telnet0 set type_ Telnet

#Setup a FTP Application over TCP connection
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 1.0 "$ftp1 start"
$ns at 10.0 "$ftp1 stop"

#=====

#Termination #=====

#Define a 'finish' procedure
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace close $tracefile close $namfile
    exec nam telnet.nam & exit 0
```

```
}  
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"  
$ns at $val(stop) "finish"  
$ns at $val(stop) "puts \"done\" ; $ns halt"  
$ns run
```

AVG_TPUT.AWK:

```
BEGIN {  
    total_recv = 0      # Total received data (in bytes)  
    startTime = 0       # Start time  
    stopTime = 0        # Stop time  
    prevTime = 0        # Previous time for calculating time  
    intervals  
    tic = 0.1           # Time threshold for periodic reporting  
    pktCount = 0        # Total received packet count  
}  
  
{  
    # Trace line format: normal  
    if ($2 != "-t") {  
        event = $1  
        time = $2  
        if (event == "+" || event == "-") {  
            node_id = $ }  
        if (event == "r" || event == "d") {  
            node_id = $4}  
        flow_id = $8  
        pkt_id = $12  
        pkt_size = $6  
        flow_t = $5  
        level = "AGT"}  
    # Trace line format: new
```

```
    if ($2 == "-t") {
        event = $1
        time = $3
        node_id = $5
        flow_id = $39
        pkt_id = $41
        pkt_size = $37
        flow_t = $45
        level = $19
    }

    # Init prevTime to the first packet receive time
    if (prevTime == 0) {
        prevTime = time
        startTime = time # Set start time to the first packet time }
    # Calculate total received packets' size and count received packets
    if (level == "AGT" && event == "r") {
        total_recv += pkt_size
        pktCount++
    }

    # Update time interval
    stopTime = time

    # Update the time and report average throughput
    total_time = stopTime - startTime

    if (total_time >= tic) {
        # Convert total received data to kilobits and calculate
        throughput (kbps)
        avg_throughput = (total_recv * 8) / (total_time * 1000)
        # Print output in the desired format
```

```

        printf("startTime: %g stopTime: %g receivedPkts:
%d\navgTput[kbps]: %f\n", startTime, stopTime, pktCount,
avg_throughput)

        total_recv = 0      # Reset total received data for next period
        pktCount = 0       # Reset packet count for next period
        startTime = stopTime # Set the new start time for the next
period
    }
    prevTime = time
}
END {
    # Final throughput report after the last period
    if (total_time > 0) {
        avg_throughput = (total_recv * 8) / (total_time * 1000)
        printf("\nstartTime: %g stopTime: %g receivedPkts:
%d\navgTput[kbps]: %f\n", startTime, stopTime, pktCount,
avg_throughput)}
    }
}

```

INSTNT_TPUT.AWK:

```

BEGIN {
    recv = 0
    currTime = prevTime = 0
    tic = 0.1
}
{
    # Trace line format: normal
    if ($2 != "-t") {
        event = $1
        time = $2
        if (event == "+" || event == "-") {
            node_id = $3

```



```
    }
    if (event == "r" || event == "d") {
        node_id = $4
    }
    flow_id = $8
    pkt_id = $12
    pkt_size = $6
    flow_t = $5
    level = "AGT"
}
# Trace line format: new
if ($2 == "-t") {
    event = $1
    time = $3
    node_id = $5
    flow_id = $39
    pkt_id = $41
    pkt_size = $37
    flow_t = $45
    level = $19
}
# Init prevTime to the first packet recv time
if (prevTime == 0) {
    prevTime = time
}
# Calculate total received packets' size
if (level == "AGT" && event == "r") {
    # Store received packet's size
    recv += pkt_size
}
```

```

    # This 'if' is introduced to obtain clearer plots from the output
    of this script
    currTime += (time - prevTime)
    if (currTime >= tic) {
        printf("%15g %18g\n", time, (recv / currTime) * (8 / 1000))
        recv = 0
        currTime = 0
    }
    prevTime = time}
END {
    printf("\n\n")
}

```

PKT.AWK:

```

BEGIN {
    total_recv = 0          # Total received data (in bytes)
    total_gen = 0           # Total generated packets (in bytes)
    total_drop = 0          # Total dropped packets
    startTime = 0           # Start time
    stopTime = 0            # Stop time
    prevTime = 0            # Previous time for calculating time
    intervals
    tic = 0.1               # Time threshold for periodic reporting
    pktCount = 0            # Total received packet count
    genPktCount = 0         # Total generated packet count
}
{
    # Trace line format: normal
    if ($2 != "-t") {
        event = $1
    }
}

```

```
    time = $2
    if (event == "+" || event == "-") {
        node_id = $3
    }
    if (event == "r" || event == "d") {
        node_id = $4
    }
    flow_id = $8
    pkt_id = $12
    pkt_size = $6
    flow_t = $5
    level = "AGT"
}
# Trace line format: new
if ($2 == "-t") {
    event = $1
    time = $3
    node_id = $5
    flow_id = $39
    pkt_id = $41
    pkt_size = $37
    flow_t = $45
    level = $19
}
# Init prevTime to the first packet receive time
if (prevTime == 0) {
    prevTime = time
    startTime = time # Set start time to the first packet time
}
# Calculate total received packets' size and count received packets
```

```
    if (level == "AGT" && event == "r") {
        total_recv += pkt_size
        pktCount++
    }
    # Calculate generated packets (assuming "+" event means generated)
    if (event == "+" || event == "-") {
        genPktCount++
    }
    # Update dropped packets count (assuming "d" event means dropped)
    if (event == "d") {
        total_drop++ }
    # Update time interval
    stopTime = time
    total_time = stopTime - startTime
    # Report periodic output
    if (total_time >= tic) {
        # Convert total received data to kilobits and calculate
        throughput (kbps)
        avg_throughput = (total_recv * 8) / (total_time * 1000)
        printf("%f\n", avg_throughput)
        total_recv = 0      # Reset total received data for next period
        pktCount = 0        # Reset packet count for next period
        startTime = stopTime # Set the new start time for the next
        period}
        prevTime = time}
    END {
        # Final throughput report after the last period
        if (total_time > 0) {
            avg_throughput = (total_recv * 8) / (total_time * 1000)
            printf("\n%f\n", avg_throughput)}
```

```
# Output the final statistics
print "GeneratedPackets = " genPktCount
print "ReceivedPackets = " pktCount
# Calculate Packet Delivery Ratio
if (genPktCount > 0) {
    pdr = (pktCount / genPktCount) * 100
    printf("Packet Delivery Ratio = %.2f%%\n", pdr)
} else {
    print "Packet Delivery Ratio = 0.00%"
}
print "Total Dropped Packets = " total_drop
}
```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

```
gedit exp1.tcl
```

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

```
ns exp1.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- For Further Analysis We Need .Awk Script Files. Copy These Files To The Same Folder As The NS2 Script.
- The .awk Files Can Be Executed By Using The Following Command:

```
awk -f filename.awk tracefile.tr
```
- The Process Is Repeated For The Files Avg_Tput.awk, Instnt_Tput.awk And Pkt.awk.
- The Outputs Are Observed And Readings Are Noted Down.

OUTPUTS:

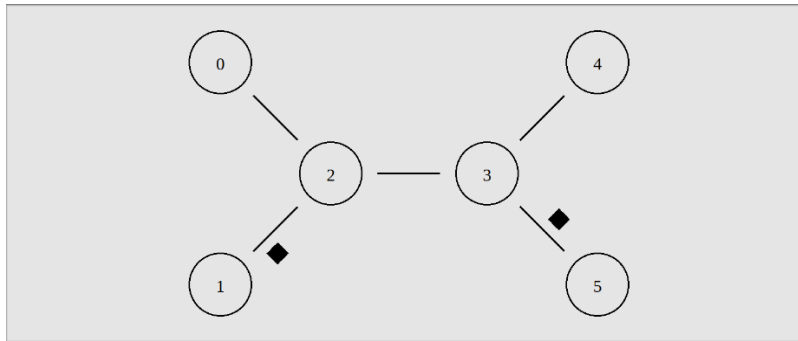


Figure 1 NAM Output

```

avgTput[kbps]: 25031.836995
startTime: 10.0565 stopTime: 10.1589 receivedPkts: 220
avgTput[kbps]: 14751.584643
startTime: 10.1589 stopTime: 10.2595 receivedPkts: 200
avgTput[kbps]: 10294.936004
startTime: 10.2595 stopTime: 10.3601 receivedPkts: 200
avgTput[kbps]: 12456.224132
startTime: 10.3601 stopTime: 10.4608 receivedPkts: 200
avgTput[kbps]: 14927.743370

startTime: 10.4608 stopTime: 10.4933 receivedPkts: 79
avgTput[kbps]: 8170.003080
ubuntu@ubuntu:~/Desktop/dcn$

```

Figure 2 Avg_Tput.awk

```

20470.625646
25031.836995
14751.584643
10294.936004
12456.224132
14927.743370

8170.003080
GeneratedPackets = 72104
ReceivedPackets = 79
Packet Delivery Ratio = 0.11%
Total Dropped Packets = 0
ubuntu@ubuntu:~/Desktop/dcn$

```

Figure 3 Pkt.awk

```
ubuntu@ubuntu:~/Desktop/dcn$ gawk -f Instnt_Tput.awk telnet.tr
1.10039      1514.06
1.20102      5130.77
1.30154      15926.5
1.40229      29830.1
1.50292      20470.6
1.60343      25031.8
1.70418      29855.5
1.80482      20470.6
1.90533      25031.8
2.00608      29855.5
2.10671      20470.6
2.20722      25031.8
2.30798      29855.5
2.40861      20470.6
2.50912      25031.8
2.60987      29855.5
```

Figure 4 Instnt_Tput.awk

RESULT:

Simulation of TELNET and FTP between N Sources and N Sinks is done using NS2 Software.

EXPERIMENT 2**EFFECT OF VARIOUS QUEUEING DISCIPLINES
(RED /WEIGHTED RED / ADAPTIVE RED) ON
NETWORK PERFORMANCE**

AIM: To Study the Effect of Various Queuing Disciplines (RED /Weighted RED /Adaptive RED) On Network Performance

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:

Queue management techniques help control network congestion by regulating how packets are queued and dropped. **Random Early Detection (RED)** is a widely used approach in queue management, and it has variants like **Weighted RED** and **Adaptive RED** to improve performance.

1. Random Early Detection (RED)

RED is an active queue management algorithm that prevents congestion by **randomly dropping packets before the queue is full**. It works by monitoring the average queue size and dropping packets probabilistically based on threshold values.

2. Weighted RED (WRED)

WRED extends RED by assigning different **drop probabilities** to different types of traffic based on priority or Quality of Service (QoS) settings. Higher-priority traffic has a lower chance of being dropped compared to lower-priority traffic. It is used in **differentiated services (DiffServ)** networks.

3. Adaptive RED (ARED)

Adaptive RED dynamically adjusts RED parameters based on network conditions to optimize performance. Instead of using fixed threshold values for packet dropping, ARED automatically **tunes the drop probability** to maintain a stable queue length. It reduces manual tuning of RED parameters and adapts to changing traffic patterns for better congestion control.

PROGRAM: RED

```
#      Simulation parameters setup
set val(stop)      10.0  ;# time of simulation end

#=====

#      Initialization
#Create a ns simulator
set ns [new Simulator]
#Open the NS trace file
set tracefile [open queue_red.tr w]
$ns trace-all $tracefile
#Open the NAM trace file
set namfile [open queue_red.nam w]
$ns namtrace-all $namfile
#=====

#      Nodes Definition
#Create 7 nodes
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]
#=====

#      Links Definition
#Createlinks between nodes
$ns duplex-link $n0 $n3 100.0Mb 10ms RED
$ns queue-limit $n0 $n3 50
$ns duplex-link $n4 $n1 100.0Mb 10ms RED
```

```
$ns queue-limit $n4 $n1 50
$ns duplex-link $n6 $n5 100.0Mb 10ms RED
$ns queue-limit $n6 $n5 50
$ns duplex-link $n4 $n6 100.0Mb 10ms RED
$ns queue-limit $n4 $n6 50
$ns duplex-link $n5 $n2 100.0Mb 10ms RED
$ns queue-limit $n5 $n2 50
$ns duplex-link $n0 $n2 100.0Mb 10ms RED
$ns queue-limit $n0 $n2 50
$ns duplex-link $n2 $n1 100.0Mb 10ms RED
$ns queue-limit $n2 $n1 50
$ns duplex-link $n1 $n5 100.0Mb 10ms RED
$ns queue-limit $n1 $n5 50
$ns duplex-link $n3 $n4 100.0Mb 10ms RED
$ns queue-limit $n3 $n4 50
$ns duplex-link $n3 $n1 100.0Mb 10ms RED
$ns queue-limit $n3 $n1 50

#Give node position (for NAM)
$ns duplex-link-op $n0 $n3 orient right-up
$ns duplex-link-op $n4 $n1 orient left-down
$ns duplex-link-op $n6 $n5 orient left-down
$ns duplex-link-op $n4 $n6 orient right-down
$ns duplex-link-op $n5 $n2 orient left

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n2 $n1 orient right-up
$ns duplex-link-op $n1 $n5 orient right-down
$ns duplex-link-op $n3 $n4 orient right
$ns duplex-link-op $n3 $n1 orient right-down
```

```
#    Agents Definition
#Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink2 [new Agent/TCPSink]
$ns attach-agent $n4 $sink2
$ns connect $tcp0 $sink2
$tcp0 set packetSize_ 1500

#Setup a TCP/FullTcp/Tahoe connection
set tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1
set sink3 [new Agent/TCPSink]
$ns attach-agent $n6 $sink3
$ns connect $tcp1 $sink3
$tcp1 set packetSize_ 1500
#=====
#    Applications Definition
#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 1.0 "$ftp0 start"
$ns at 10.0 "$ftp0 stop"
#Setup a FTP Application over TCP/FullTcp/Tahoe connection
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 1.0 "$ftp1 start"
$ns at 10.0 "$ftp1 stop"
```

```

#      Termination
#Define a 'finish' procedure
proc finish {} {
  global ns tracefile namfile
  $ns flush-trace
  close $tracefile
  close $namfile
  exec nam queue_red.nam &
  exit 0
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run

```

PROGRAM: WRED

```

#      Simulation parameters setup #=====
set val(stop)      10.0  ;# time of simulation end

#=====

#      Initialization #=====

#Create a ns simulator
set ns [new Simulator]

#Open the NS trace file
set tracefile [open queue_wred.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open queue_wred.nam w]

```

```
$ns namtrace-all $namfile

#=====

#    Nodes Definition #=====

#Create 7 nodes
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]

#=====

#    Links Definition #=====

#Create links with Weighted RED (WRED)
$ns duplex-link $n0 $n3 100.0Mb 10ms RED/Queue
$ns queue-limit $n0 $n3 50
$ns duplex-link $n4 $n1 100.0Mb 10ms RED/Queue
$ns queue-limit $n4 $n1 50
$ns duplex-link $n6 $n5 100.0Mb 10ms RED/Queue
$ns queue-limit $n6 $n5 50
$ns duplex-link $n4 $n6 100.0Mb 10ms RED/Queue
$ns queue-limit $n4 $n6 50
$ns duplex-link $n5 $n2 100.0Mb 10ms RED/Queue
$ns queue-limit $n5 $n2 50
$ns duplex-link $n0 $n2 100.0Mb 10ms RED/Queue
$ns queue-limit $n0 $n2 50
$ns duplex-link $n2 $n1 100.0Mb 10ms RED/Queue
$ns queue-limit $n2 $n1 50
```

```

$ns duplex-link $n1 $n5 100.0Mb 10ms RED/Queue
$ns queue-limit $n1 $n5 50
$ns duplex-link $n3 $n4 100.0Mb 10ms RED/Queue
$ns queue-limit $n3 $n4 50
$ns duplex-link $n3 $n1 100.0Mb 10ms RED/Queue
$ns queue-limit $n3 $n1 50

```

Weighted RED Configuration

```

Queue/RED set gentle_ true
Queue/RED set thresh_ 20
Queue/RED set maxthresh_ 40
Queue/RED set weight_ 0.002
Queue/RED set linterm_ 10
Queue/RED set q_weight_ 0.003

```

#Give node position (for NAM)

```

$ns duplex-link-op $n0 $n3 orient right-up
$ns duplex-link-op $n4 $n1 orient left-down
$ns duplex-link-op $n6 $n5 orient left-down
$ns duplex-link-op $n4 $n6 orient right-down
$ns duplex-link-op $n5 $n2 orient left

```

```

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n2 $n1 orient right-up
$ns duplex-link-op $n1 $n5 orient right-down
$ns duplex-link-op $n3 $n4 orient right
$ns duplex-link-op $n3 $n1 orient right-down

```

```
#=====
```

```
# Agents Definition #=====
```

```
#Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink2 [new Agent/TCPSink]
$ns attach-agent $n4 $sink2
$ns connect $tcp0 $sink2
$tcp0 set packetSize_ 1500

#Setup a TCP/FullTcp/Tahoe connection
set tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1
set sink3 [new Agent/TCPSink]
$ns attach-agent $n6 $sink3
$ns connect $tcp1 $sink3
$tcp1 set packetSize_ 1500

#=====
#   Applications Definition #=====
#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 1.0 "$ftp0 start"
$ns at 10.0 "$ftp0 stop"

#Setup a FTP Application over TCP/FullTcp/Tahoe connection
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 1.0 "$ftp1 start"
$ns at 10.0 "$ftp1 stop"
```

```

#=====

#    Termination #=====

#Define a 'finish' procedure
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam queue_wred.nam &
    exit 0
}

$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run

```

PROGRAM: ARED

```

#    Simulation parameters setup #=====

set val(stop)    10.0    ;# time of simulation end


#=====

#    Initialization #=====

#Create a ns simulator
set ns [new Simulator]


#Open the NS trace file
set tracefile [open queue_ared.tr w]
$ns trace-all $tracefile


#Open the NAM trace file

```



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

#=====

#   Nodes Definition #=====
#Create 7 nodes
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]

#=====

#   Links Definition #=====
#Create links with Adaptive RED (ARED)
$ns duplex-link $n0 $n3 100.0Mb 10ms RED/ARED
$ns queue-limit $n0 $n3 50
$ns duplex-link $n4 $n1 100.0Mb 10ms RED/ARED
$ns queue-limit $n4 $n1 50
$ns duplex-link $n6 $n5 100.0Mb 10ms RED/ARED
$ns queue-limit $n6 $n5 50
$ns duplex-link $n4 $n6 100.0Mb 10ms RED/ARED
$ns queue-limit $n4 $n6 50
$ns duplex-link $n5 $n2 100.0Mb 10ms RED/ARED
$ns queue-limit $n5 $n2 50
$ns duplex-link $n0 $n2 100.0Mb 10ms RED/ARED
$ns queue-limit $n0 $n2 50
$ns duplex-link $n2 $n1 100.0Mb 10ms RED/ARED
```

```
$ns queue-limit $n2 $n1 50
$ns duplex-link $n1 $n5 100.0Mb 10ms RED/ARED
$ns queue-limit $n1 $n5 50
$ns duplex-link $n3 $n4 100.0Mb 10ms RED/ARED
$ns queue-limit $n3 $n4 50
$ns duplex-link $n3 $n1 100.0Mb 10ms RED/ARED
$ns queue-limit $n3 $n1 50
```

```
# Adaptive RED Configuration
Queue/RED set adaptive_ true
Queue/RED set thresh_ 20
Queue/RED set maxthresh_ 40
Queue/RED set q_weight_ 0.002
Queue/RED set linterm_ 10
Queue/RED set gentle_ true
Queue/RED set adaptive_max_p_ 0.02
Queue/RED set adaptive_incr_ 0.01
Queue/RED set adaptive_decr_ 0.00001
```

```
#Give node position (for NAM)
$ns duplex-link-op $n0 $n3 orient right-up
$ns duplex-link-op $n4 $n1 orient left-down
$ns duplex-link-op $n6 $n5 orient left-down
$ns duplex-link-op $n4 $n6 orient right-down
$ns duplex-link-op $n5 $n2 orient left
```

```
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n2 $n1 orient right-up
$ns duplex-link-op $n1 $n5 orient right-down
$ns duplex-link-op $n3 $n4 orient right
```

```
$ns duplex-link-op $n3 $n1 orient right-down

#=====

#    Agents Definition #=====

#Setup a TCP connection
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink2 [new Agent/TCPSink]
$ns attach-agent $n4 $sink2
$ns connect $tcp0 $sink2
$tcp0 set packetSize_ 1500

#Setup a TCP/FullTcp/Tahoe connection
set tcp1 [new Agent/TCP]
$ns attach-agent $n2 $tcp1
set sink3 [new Agent/TCPSink]
$ns attach-agent $n6 $sink3
$ns connect $tcp1 $sink3
$tcp1 set packetSize_ 1500

#=====

#    Applications Definition #=====

#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 1.0 "$ftp0 start"
$ns at 10.0 "$ftp0 stop"

#Setup a FTP Application over TCP/FullTcp/Tahoe connection
set ftp1 [new Application/FTP]
```

```

$ftp1 attach-agent $tcp1
$ns at 1.0 "$ftp1 start"
$ns at 10.0 "$ftp1 stop"

#=====
#      Termination #=====
#Define a 'finish' procedure
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam queue_ared.nam &
    exit 0
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run

```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

```
gedit exp2.tcl
```

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

```
ns exp2.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- The Outputs Are Observed And Readings Are Noted Down.

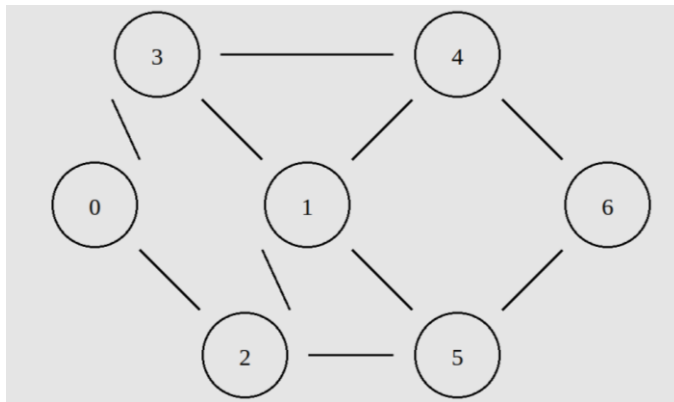
OUTPUTS:

Figure 5 NAM Output

```

Open ▾ [icon]
+ 1 0 3 tcp 40 ----- 0 0.0 4.0 0 0
- 1 0 3 tcp 40 ----- 0 0.0 4.0 0 0
+ 1 2 5 tcp 40 ----- 0 2.0 6.0 0 1
- 1 2 5 tcp 40 ----- 0 2.0 6.0 0 1
r 1.010003 0 3 tcp 40 ----- 0 0.0 4.0 0 0
+ 1.010003 3 4 tcp 40 ----- 0 0.0 4.0 0 0
- 1.010003 3 4 tcp 40 ----- 0 0.0 4.0 0 0
r 1.010003 2 5 tcp 40 ----- 0 2.0 6.0 0 1
+ 1.010003 5 6 tcp 40 ----- 0 2.0 6.0 0 1
- 1.010003 5 6 tcp 40 ----- 0 2.0 6.0 0 1
r 1.020006 3 4 tcp 40 ----- 0 0.0 4.0 0 0
+ 1.020006 4 3 ack 40 ----- 0 4.0 0.0 0 2
- 1.020006 4 3 ack 40 ----- 0 4.0 0.0 0 2
r 1.020006 5 6 tcp 40 ----- 0 2.0 6.0 0 1
+ 1.020006 6 5 ack 40 ----- 0 6.0 2.0 0 3
- 1.020006 6 5 ack 40 ----- 0 6.0 2.0 0 3
r 1.03001 4 3 ack 40 ----- 0 4.0 0.0 0 2
+ 1.03001 3 0 ack 40 ----- 0 4.0 0.0 0 2
- 1.03001 3 0 ack 40 ----- 0 4.0 0.0 0 2
r 1.03001 6 5 ack 40 ----- 0 6.0 2.0 0 3
+ 1.03001 5 2 ack 40 ----- 0 6.0 2.0 0 3
- 1.03001 5 2 ack 40 ----- 0 6.0 2.0 0 3

```

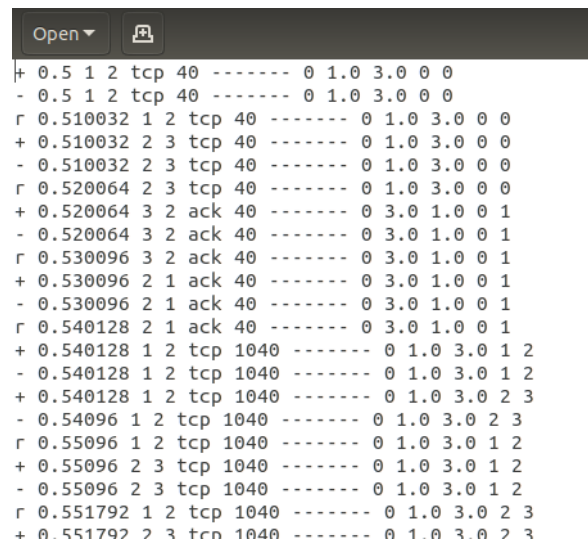
Figure 6 Tracefile:RED

```

Open ▾ [icon]
+ 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
- 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
r 0.510032 1 2 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
- 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
r 0.520064 2 3 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
- 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
r 0.530096 3 2 ack 40 ----- 0 3.0 1.0 0 1
+ 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1
- 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1
r 0.540128 2 1 ack 40 ----- 0 3.0 1.0 0 1
+ 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 1 2
- 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 1 2
+ 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 2 3
- 0.54096 1 2 tcp 1040 ----- 0 1.0 3.0 2 3
r 0.55096 1 2 tcp 1040 ----- 0 1.0 3.0 1 2
+ 0.55096 2 3 tcp 1040 ----- 0 1.0 3.0 1 2
- 0.55096 2 3 tcp 1040 ----- 0 1.0 3.0 1 2
r 0.551792 1 2 tcp 1040 ----- 0 1.0 3.0 2 3
+ 0.551792 2 3 tcp 1040 ----- 0 1.0 3.0 2 3

```

Figure 7 Tracefile:WRED



```

Open ▾
+ 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
- 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
r 0.510032 1 2 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
- 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
r 0.520064 2 3 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
- 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
r 0.530096 3 2 ack 40 ----- 0 3.0 1.0 0 1
+ 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1
- 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1
r 0.540128 2 1 ack 40 ----- 0 3.0 1.0 0 1
+ 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 1 2
- 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 1 2
+ 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 2 3
- 0.54096 1 2 tcp 1040 ----- 0 1.0 3.0 2 3
r 0.55096 1 2 tcp 1040 ----- 0 1.0 3.0 1 2
+ 0.55096 2 3 tcp 1040 ----- 0 1.0 3.0 1 2
- 0.55096 2 3 tcp 1040 ----- 0 1.0 3.0 1 2
r 0.551792 1 2 tcp 1040 ----- 0 1.0 3.0 2 3
+ 0.551792 2 3 tcp 1040 ----- 0 1.0 3.0 2 3

```

Figure 8 Tracefile:ARED

RESULT:

The Effect of Various Queuing Disciplines (RED /Weighted RED /Adaptive RED) On Network Performance is Observed Using NS2 Software.

EXPERIMENT 3**SIMULATION OF FTP, HTTP, DBMS ACCESS IN NETWORKS**

AIM: To simulate http, ftp and DBMS access in networks

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:

1. HTTP (Hypertext Transfer Protocol)

- **Purpose:** HTTP is a protocol used for transferring hypertext (web pages) from a web server to a web browser (client).
- **Port Number:** 80 (Default)
- **Working:**
 - The client sends a **HTTP request** (GET, POST, PUT and DELETE) to the server.
 - The server responds with a **HTTP response** containing the requested resource (web page, image, etc.).
- **Stateless Protocol:** Each request-response is independent, meaning no information is retained between requests.
- **Secure Version:** HTTPS (uses SSL/TLS encryption)

2. FTP (File Transfer Protocol)

- **Purpose:** FTP is used for transferring files between a **client** and a **server** in a network.
- **Port Number:**
 - **21:** Control connection (for commands)
 - **20:** Data connection (for file transfer)
- **Modes:**
 - **Active Mode:** Server actively establishes a data connection.
 - **Passive Mode:** Client requests a data connection from the server.
- **Commands:**
 - PUT → Upload files from client to server.
 - GET → Download files from server to client.

3. DBMS Access in Networks

- **Purpose:** DBMS (Database Management System) access in networks allows remote applications or users to access and manipulate data stored in a centralized database server.
- **Protocol Used:**
 - **ODBC (Open Database Connectivity)**
 - **JDBC (Java Database Connectivity)**
- **Port Numbers:**
 - **3306:** MySQL Database
 - **1433:** MS-SQL Server
 - **1521:** Oracle Database

PROGRAM:

```
#=====

# NS2 Simulation for HTTP, FTP, DBMS Access

#=====

# Simulation Parameters

set val(stop) 10.0 ;# Simulation Stop Time

# Initialize the Simulator

set ns [new Simulator]

# Trace Files

set tracefile [open network_access.tr w]

$ns trace-all $tracefile

set namfile [open network_access.nam w]

$ns namtrace-all $namfile

# Create Nodes

set n0 [$ns node] ;# HTTP Client

set n1 [$ns node] ;# HTTP Server

set n2 [$ns node] ;# FTP Client
```



```
set n3 [$ns node] ;# FTP Server

set n4 [$ns node] ;# DBMS Client

set n5 [$ns node] ;# DBMS Server

#=====

# Link Creation

#=====

# Link between HTTP Client and HTTP Server

$ns duplex-link $n0 $n1 100Mb 10ms DropTail

# Link between FTP Client and FTP Server

$ns duplex-link $n2 $n3 100Mb 10ms DropTail

# Link between DBMS Client and DBMS Server

$ns duplex-link $n4 $n5 100Mb 10ms DropTail

# Set Queue Size

$ns queue-limit $n0 $n1 50

$ns queue-limit $n2 $n3 50

$ns queue-limit $n4 $n5 50

#=====

# HTTP Simulation Setup

#=====

# Create TCP Agent for HTTP Traffic

set tcp_http [new Agent/TCP]

$ns attach-agent $n0 $tcp_http

# Create Sink Agent

set sink_http [new Agent/TCPSink]
```

```
$ns attach-agent $n1 $sink_http

# Connect the Agents

$ns connect $tcp_http $sink_http

$tcp_http set packetSize_ 512

# Create HTTP Traffic (Application Layer)

set http [new Application/Traffic/Exponential]

$http attach-agent $tcp_http

$http set rate_ 2Mb

$http set burst_time_ 2.0

$http set idle_time_ 1.0

# Start HTTP Traffic

$ns at 1.0 "$http start"

$ns at 9.0 "$http stop"

#=====

# FTP Simulation Setup

#=====

# Create TCP Agent for FTP Traffic

set tcp_ftp [new Agent/TCP]

$ns attach-agent $n2 $tcp_ftp

# Create Sink Agent

set sink_ftp [new Agent/TCPSink]

$ns attach-agent $n3 $sink_ftp

# Connect the Agents

$ns connect $tcp_ftp $sink_ftp
```

```
$tcp_ftp set packetSize_ 1500

# Create FTP Application

set ftp [new Application/FTP]

$ftp attach-agent $tcp_ftp

# Start FTP Traffic

$ns at 2.0 "$ftp start"

$ns at 8.0 "$ftp stop"

#=====

# DBMS Access Simulation Setup

#=====

# Create TCP Agent for DBMS Access

set tcp_db [new Agent/TCP]

$ns attach-agent $n4 $tcp_db

# Create Sink Agent

set sink_db [new Agent/TCPSink]

$ns attach-agent $n5 $sink_db

# Connect the Agents

$ns connect $tcp_db $sink_db

$tcp_db set packetSize_ 1000

# Simulate Database Access with CBR Traffic (DBMS Query)

set db_query [new Application/Traffic/CBR]

$db_query attach-agent $tcp_db

$db_query set rate_ 1Mb

$db_query set packetSize_ 1000
```

```

$db_query set interval_ 0.01

# Start Database Access Traffic

$ns at 3.0 "$db_query start"

$ns at 7.0 "$db_query stop"

#=====

# Termination Setup

#=====

proc finish {} {

    global ns tracefile namfile

    $ns flush-trace

    close $tracefile

    close $namfile

    exec nam network_access.nam &

    exit 0

}

$ns at $val(stop) "finish"

$ns at $val(stop) "puts \"Simulation Complete\" ; $ns halt"

# Run the Simulation

$ns run

```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

`gedit exp3.tcl`

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

```
ns exp3.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- The Outputs Are Observed And Readings Are Noted Down.

OUTPUTS:

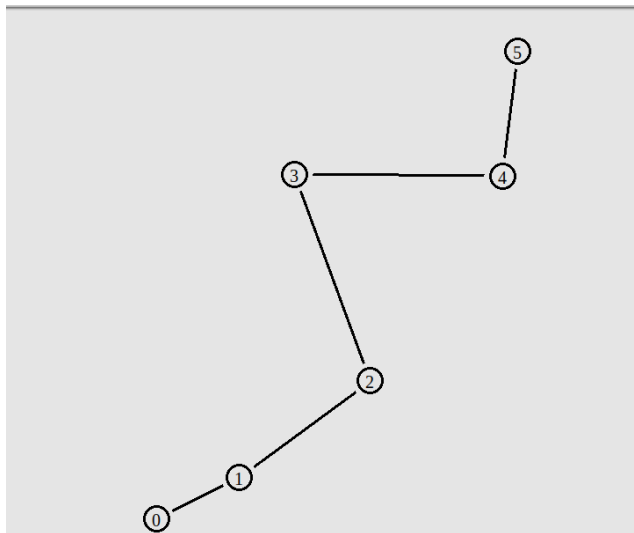


Figure 9: NAM Output

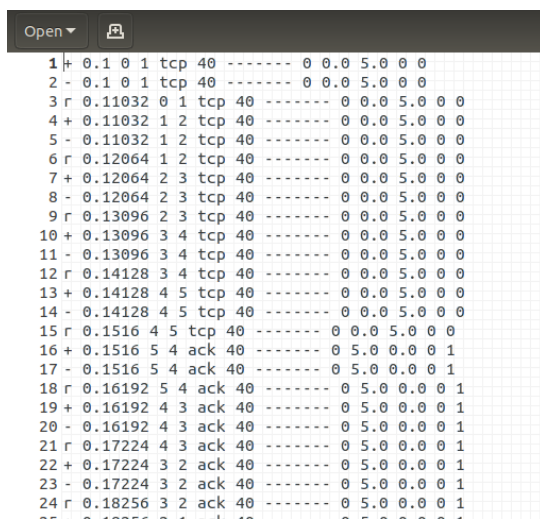


Figure 10 Tracefile

RESULT:

Simulation of HTTP, FTP, DBMS Access in Networks is Done Using NS2 Software.

EXPERIMENT 4**SIMULATION OF SLIDING WINDOW PROTOCOL**

AIM: To simulate sliding window protocol.

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:

The **Sliding Window Protocol (SWP)** is a **flow control protocol** used in **Data Link Layer (Layer 2)** and **Transport Layer (Layer 4)** to ensure **efficient and reliable** transmission of data between two devices over a network.

It is mainly used in:

- **TCP (Transmission Control Protocol).**
- **Data Link Layer (for error and flow control).**
- A "**window**" refers to the number of **packets (frames)** that can be sent before **waiting for an acknowledgment (ACK)**.
- As **ACKs** are received, the window **slides** forward to allow the sender to transmit more packets.

PROGRAM:

```
# Define a network simulator
set ns [new Simulator]

# Create a trace file
set tracefile [open "output.tr" w]
$ns trace-all $tracefile

# Create nodes
set sender [$ns node]
set receiver [$ns node]

# Create a link between sender and receiver
$ns duplex-link $sender $receiver 1Mb 10ms DropTail

# Define the traffic type (TCP)
set tcp [new Agent/TCP]
$ns attach-agent $sender $tcp

# Define TCP sink
set sink [new Agent/TCPSink]
$ns attach-agent $receiver $sink

# Connect TCP agent and sink
$ns connect $tcp $sink

# Set sliding window size
$tcp set window_ 4 ;# Example: Window size = 4 frames

# Define traffic source
set ftp [new Application/FTP]
$ftp attach-agent $tcp

# Schedule events
$ns at 0.1 "$ftp start"
$ns at 5.0 "$ftp stop"
$ns at 5.1 "finish"

# Define finish procedure
```



```
proc finish {} {  
    global ns tracefile  
    $ns flush-trace  
    close $tracefile  
    exit 0  
}  
# Run the simulation  
$ns run
```

ANALYZE.AWK:

```
BEGIN {  
  
    sent = 0; received = 0; dropped = 0; total_delay = 0;}  
  
# Count sent packets  
  
$1 == "+" { sent++; send_time[$11] = $2; }  
  
# Count received packets and calculate delay  
  
$1 == "r" {  
  
    received++;  
  
    if ($11 in send_time) {  
  
        total_delay += ($2 - send_time[$11]);  
  
        delete send_time[$11];  
  
    }  
  
}  
  
# Count dropped packets  
  
$1 == "d" { dropped++; }  
  
END {  
  
    throughput = (received * 1024 * 8) / 5.0; # Assuming 5s simulation time
```

```

    avg_delay = (received > 0) ? total_delay / received : 0;

    drop_rate = (sent > 0) ? (dropped / sent) * 100 : 0;

    printf("Throughput: %.2f bps\n", throughput);

    printf("Average Delay: %.2f ms\n", avg_delay * 1000);

    printf("Packet Drop Rate: %.2f%%\n", drop_rate);

}

```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

```
gedit exp4.tcl
```

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

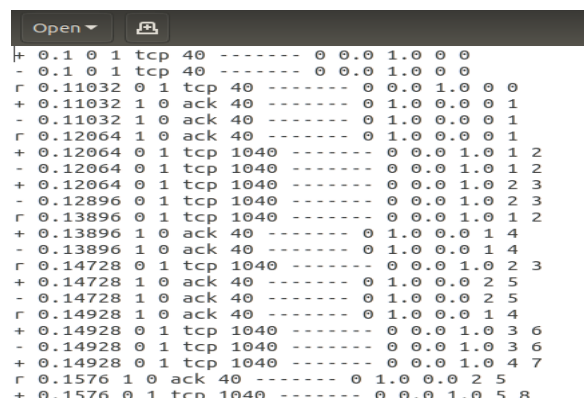
```
ns exp4.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- For Further Analysis We Need .Awk Script Files. Copy These Files To The Same Folder As The NS2 Script.
- The .awk Files Can Be Executed By Using The Following Command:

```
awk -f analyze.awk tracefile.tr
```

- The Outputs Are Observed And Readings Are Noted Down.

OUTPUTS:

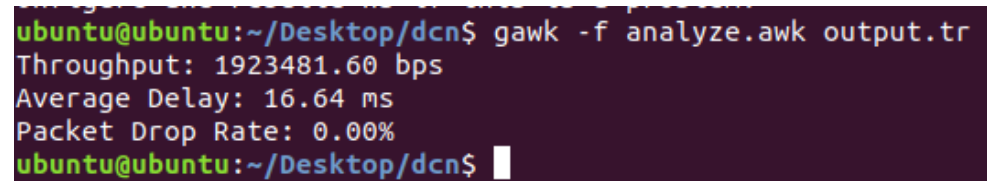


```

+ 0.1 0 1 tcp 40 ----- 0 0.0 1.0 0 0
- 0.1 0 1 tcp 40 ----- 0 0.0 1.0 0 0
r 0.11032 0 1 tcp 40 ----- 0 0.0 1.0 0 0
+ 0.11032 1 0 ack 40 ----- 0 1.0 0.0 0 1
- 0.11032 1 0 ack 40 ----- 0 1.0 0.0 0 1
r 0.12064 1 0 ack 40 ----- 0 1.0 0.0 0 1
+ 0.12064 0 1 tcp 1040 ----- 0 0.0 1.0 1 2
- 0.12064 0 1 tcp 1040 ----- 0 0.0 1.0 1 2
+ 0.12064 0 1 tcp 1040 ----- 0 0.0 1.0 2 3
- 0.12896 0 1 tcp 1040 ----- 0 0.0 1.0 2 3
r 0.13896 0 1 tcp 1040 ----- 0 0.0 1.0 1 2
+ 0.13896 1 0 ack 40 ----- 0 1.0 0.0 1 4
- 0.13896 1 0 ack 40 ----- 0 1.0 0.0 1 4
r 0.14728 0 1 tcp 1040 ----- 0 0.0 1.0 2 3
+ 0.14728 1 0 ack 40 ----- 0 1.0 0.0 2 5
- 0.14728 1 0 ack 40 ----- 0 1.0 0.0 2 5
r 0.14928 1 0 ack 40 ----- 0 1.0 0.0 1 4
+ 0.14928 0 1 tcp 1040 ----- 0 0.0 1.0 3 6
- 0.14928 0 1 tcp 1040 ----- 0 0.0 1.0 3 6
+ 0.14928 0 1 tcp 1040 ----- 0 0.0 1.0 4 7
r 0.1576 1 0 ack 40 ----- 0 1.0 0.0 2 5
+ 0.1576 0 1 tcp 1040 ----- 0 0.0 1.0 5 8

```

Figure 11 Tracefile

A terminal window with a dark purple background. The prompt is 'ubuntu@ubuntu:~/Desktop/dcn\$'. The command 'gawk -f analyze.awk output.tr' has been executed. The output is displayed in three lines: 'Throughput: 1923481.60 bps', 'Average Delay: 16.64 ms', and 'Packet Drop Rate: 0.00%'. The prompt is repeated on the next line with a white cursor.

```
ubuntu@ubuntu:~/Desktop/dcn$ gawk -f analyze.awk output.tr
Throughput: 1923481.60 bps
Average Delay: 16.64 ms
Packet Drop Rate: 0.00%
ubuntu@ubuntu:~/Desktop/dcn$
```

Figure 12 Analyze.awk Output

RESULT:

Simulation of Sliding Window Protocol is Done Using NS2 Software.

EXPERIMENT 5**SIMULATION OF GO BACK N, SELECTIVE REPEAT PROTOCOLS**

AIM: To simulate GoBackN and Selective Repeat Protocols using NS2.

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:

The Go-Back-N protocol is a sliding window protocol used for reliable data transfer in computer networks. It is a sender-based protocol that allows the sender to transmit multiple packets without waiting for an acknowledgement for each packet. The receiver sends a cumulative acknowledgement for a sequence of packets, indicating the last correctly received packet. If any packet is lost, the receiver sends a negative acknowledgement (NACK) for the lost packet, and the sender retransmits all the packets in the window starting from the lost packet. The sender also maintains a timer for each packet, and if an acknowledgement is not received within the timer's timeout period, the sender retransmits all packets in the window.

The Selective Repeat protocol is another the used for reliable data transfer in computer networks. It is a receiver-based protocol that allows the receiver to acknowledge each packet individually, rather than a cumulative acknowledgement of a sequence of packets. The sender sends packets in a window and waits for acknowledgements for each packet in the window. If a packet is lost, the receiver sends a NACK for the lost packet, and the sender retransmits only that packet. The sender also maintains a timer for each packet, and if an acknowledgement is not received within the timer's timeout period, the sender retransmits only that packet.

PROGRAM:

```
# Define a network simulator

set ns [new Simulator]

# Create trace file for simulation analysis

set tracefile [open "error_model_comparison.tr" w]

$ns trace-all $tracefile

# Create nodes

set sender [$ns node]

set receiver [$ns node]

# Create a link between sender and receiver

$ns duplex-link $sender $receiver 1Mb 10ms DropTail

# Define error model (Uniform Error Model)

# This introduces packet loss in the link

set loss_model [new ErrorModel]

$loss_model set rate_ 0.05 ;# 5% packet loss

$loss_model unit pkt

$loss_model ranvar [new RandomVariable/Uniform]

$loss_model drop-target [new Agent/Null]

$ns lossmodel $loss_model $sender $receiver

# Function to configure and simulate Go-Back-N or Selective Repeat

proc simulate {protocol_type window_size simulation_time} {

    global ns sender receiver tracefile

    set tcp [new Agent/TCP]

    $tcp set class_ $protocol_type ;# Set protocol: TCP/GoBackN or
    TCP/SelectiveRepeat
```

```
$tcp set window_ $window_size ;# Configure sliding window size

set sink [new Agent/TCPSink]

$ns attach-agent $sender $tcp

$ns attach-agent $receiver $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ns at 0.1 "$ftp start"

$ns at $simulation_time "$ftp stop" }

# Simulate Go-Back-N (GBN) with varying window sizes

puts "Simulating Go-Back-N..."

simulate "GoBackN" 4 5.0 ;# GBN with window size 4

simulate "GoBackN" 8 5.0 ;# GBN with window size 8

# Simulate Selective Repeat (SR) with varying window sizes

puts "Simulating Selective Repeat..."

simulate "SelectiveRepeat" 4 5.0 ;# SR with window size 4

simulate "SelectiveRepeat" 8 5.0 ;# SR with window size 8

# Define finish procedure

proc finish {} {

    global ns tracefile

    $ns flush-trace

    close $tracefile

    exit 0 }

# Schedule simulation end
```

```
$ns at 6.0 "finish"

# Run the simulation

$ns run

ANALYZE.AWK:

BEGIN {
sent = 0; received = 0; dropped = 0; total_delay = 0;}
# Count sent packets
$1 == "+" { sent++; send_time[$11] = $2; }
# Count received packets and calculate delay
$1 == "r" {
received++;
if ($11 in send_time) {
total_delay += ($2 - send_time[$11]);
delete send_time[$11];
}
}
# Count dropped packets
$1 == "d" { dropped++; }
END {
throughput = (received * 1024 * 8) / 5.0; # Assuming 5s simulation time
avg_delay = (received > 0) ? total_delay / received : 0;
drop_rate = (sent > 0) ? (dropped / sent) * 100 : 0;
printf("Throughput: %.2f bps\n", throughput);
printf("Average Delay: %.2f ms\n", avg_delay * 1000);
printf("Packet Drop Rate: %.2f%%\n", drop_rate);
}
```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

```
gedit exp5.tcl
```

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

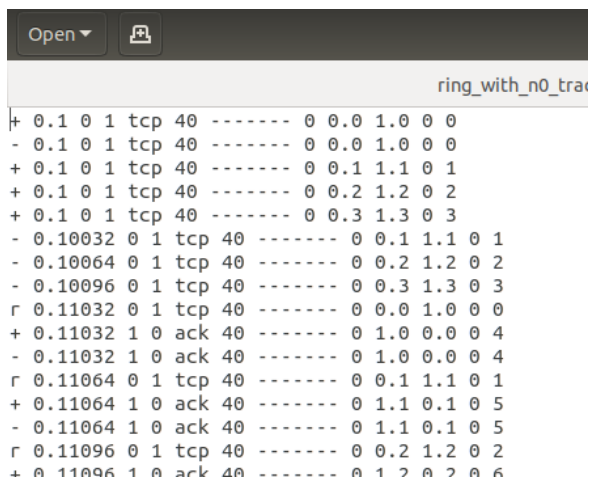
```
ns exp5.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- For Further Analysis We Need .Awk Script Files. Copy These Files To The Same Folder As The NS2 Script.

- The .awk Files Can Be Executed By Using The Following Command:

```
awk -f analyze.awk tracefile.tr
```

- The Outputs Are Observed And Readings Are Noted Down.

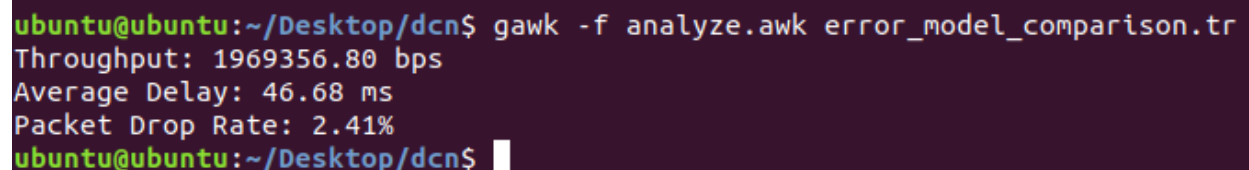
OUTPUTS:


```

+ 0.1 0 1 tcp 40 ----- 0 0.0 1.0 0 0
- 0.1 0 1 tcp 40 ----- 0 0.0 1.0 0 0
+ 0.1 0 1 tcp 40 ----- 0 0.1 1.1 0 1
+ 0.1 0 1 tcp 40 ----- 0 0.2 1.2 0 2
+ 0.1 0 1 tcp 40 ----- 0 0.3 1.3 0 3
- 0.10032 0 1 tcp 40 ----- 0 0.1 1.1 0 1
- 0.10064 0 1 tcp 40 ----- 0 0.2 1.2 0 2
- 0.10096 0 1 tcp 40 ----- 0 0.3 1.3 0 3
r 0.11032 0 1 tcp 40 ----- 0 0.0 1.0 0 0
+ 0.11032 1 0 ack 40 ----- 0 1.0 0.0 0 4
- 0.11032 1 0 ack 40 ----- 0 1.0 0.0 0 4
r 0.11064 0 1 tcp 40 ----- 0 0.1 1.1 0 1
+ 0.11064 1 0 ack 40 ----- 0 1.1 0.1 0 5
- 0.11064 1 0 ack 40 ----- 0 1.1 0.1 0 5
r 0.11096 0 1 tcp 40 ----- 0 0.2 1.2 0 2
+ 0.11096 1 0 ack 40 ----- 0 1.2 0.2 0 6

```

Figure 13 Tracefile Output



```

ubuntu@ubuntu:~/Desktop/dcn$ gawk -f analyze.awk error_model_comparison.tr
Throughput: 1969356.80 bps
Average Delay: 46.68 ms
Packet Drop Rate: 2.41%
ubuntu@ubuntu:~/Desktop/dcn$

```

Figure 14 Analyze.awk Output

RESULT:

Comparison of Go Back N Protocol and Selective Repeat Protocol is Done Using NS2 Software.

EXPERIMENT 6**COMPARISION OF CSMA/CA AND CSMA/CD
PROTOCOLS**

AIM: To study the performance of network with CSMA/CA protocol and compare with CSMA/CD protocols.

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:

CSMA/CA stands for Carrier Sense Multiple Access / Collision Avoidance is a network protocol for carrier transmission. Like CSMA/CD it is also operated in the medium access control layer. Unlike CSMA/CD (that is effective after a collision) CSMA / CA is effective before a collision. CSMA/CA reduces the chances of collisions by checking whether network channel is free or not before sending data. This is useful in wireless networks where collisions can interrupt communication. It is suitable for wireless system as it avoids collisions, by maintaining smoother connection.

CSMA/CD stands for Carrier Sense Multiple Access / Collision Detection is a network protocol for carrier transmission. It is operated in the medium access control layer. It senses that the shared channel is busy broadcasting and interrupts the broadcast until the channel is free. In CSMA/CD collision is detected by broadcast sensing from the other stations. Upon collision detection in CSMA/CD, the transmission is stopped, and a jam signal is sent by the stations and then the station waits for a random time context before retransmission. It works well in wired network like Ethernet, as it is easier to detect both the Collision and response quickly, making it ideal and efficient for wired network. It is easier to implement in wired environments, especially in networks where traffic is manageable and the chances of collision is also lower.

PROGRAM: CSMA/CA:

```
set ns [new Simulator]

#Define different colors for data flows (for NAM)

$ns color 1 Blue

$ns color 2 Red

#Open the Trace files

set file1 [open out.tr w]

set winfile [open WinFile w]

$ns trace-all $file1

#Open the NAM trace files

set file2 [open out.nam w]

$ns namtrace-all $file2

#Define a 'finish' procedure

proc finish {} {

    global ns file1 file2

    $ns flush-trace

    close $file1

    close $file2

    exec nam out.nam &

    exit 0 }

#Create six nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]
```

```
set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]

$n1 color red

$n1 shape box

#Create links between the nodes

$ns duplex-link $n0 $n2 2Mb 10ms DropTail

$ns duplex-link $n1 $n2 2Mb 10ms DropTail

$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail

$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail

set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL
Queue/DropTailMAC/Csma/Ca Channel]

#Setup a TCP connection

set tcp [new Agent/TCP/Newreno]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink/DelAck]

$ns attach-agent $n4 $sink

$ns connect $tcp $sink

$tcp set fid_ 1

$tcp set window_ 8000

$tcp set packetSize_ 552

#Setup a FTP over TCP connection

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ftp set type_ FTP
```

```
#Setup a UDP connection

set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n5 $null

$ns connect $udp $null

$udp set fid_ 2

#Setup a CBR over UDP connection

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set type_ CBR

$cbr set packet_size_ 1000

$cbr set rate_ 0.01mb

$cbr set random_ false

$ns at 0.1 "$cbr start"

$ns at 1.0 "$ftp start"

$ns at 124.0 "$ftp stop"

$ns at 124.5 "$cbr stop"

# next procedure gets two arguments: the name of the # tcp source node,
will be called here "tcp",

# and the name of output file.

proc plotWindow {tcpSource file} {

    global ns

    set time 0.1

    set now [$ns now]
```

```
set cwnd [$tcpSource set cwnd_]
set wnd [$tcpSource set window_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns at 0.1 "plotWindow $tcp $winfile"
$ns at 5 "$ns trace-annotate \"packet drop\""
# PPP
$ns at 125.0 "finish"
$ns run
```

PROGRAM: CSMA/CD:

```
set ns [new Simulator]
#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
#Open the Trace files
set file1 [open out.tr w]
set winfile [open WinFile w]
$ns trace-all $file1
#Open the NAM trace file
set file2 [open out.nam w]
$ns namtrace-all $file2
#Define a 'finish' procedure
proc finish {} {
    global ns file1 file2
    $ns flush-trace
    close $file1
    close $file2
}
```

```
exec nam out.nam
exit 0 }
#Create six nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n1 color red
$n1 shape box
#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL
Queue/DropTailMAC/Csma/Cd Channel]
#Setup a TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 8000
$tcp set packetSize_ 552
#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

```
$ftp set type_ FTP
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01mb
$cbr set random_ false
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 124.0 "$ftp stop"
$ns at 124.5 "$cbr stop"

# next procedure gets two arguments: the name of the # tcp source node,
will be called here "tcp",
# and the name of output file.
proc plotWindow {tcpSource file} {
    global ns
    set time 0.1
    set now [$ns now]
    set cwnd [$tcpSource set cwnd_]
    set wnd [$tcpSource set window_]
    puts $file "$now $cwnd"
    $ns at [expr $now+$time] "plotWindow $tcpSource $file" }
```



```
$ns at 0.1 "plotWindow $tcp $winfile"  
$ns at 5 "$ns trace-annotate \"packet drop\""  
# PPP  
$ns at 125.0 "finish"  
$ns run
```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

```
gedit exp6.tcl
```

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

```
ns exp6.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- The Outputs Are Observed And Readings Are Noted Down.

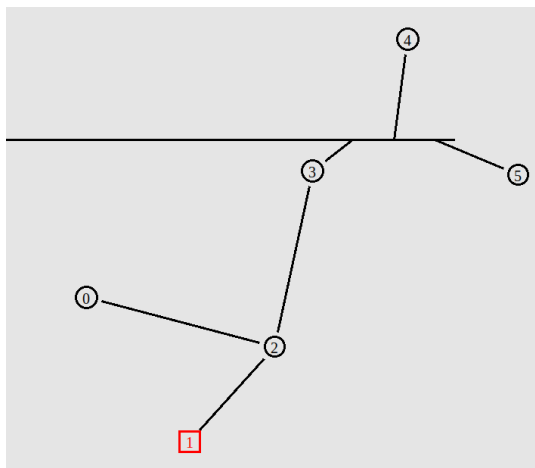
OUTPUTS:

Figure 15 NAM Output: CSMA/CA

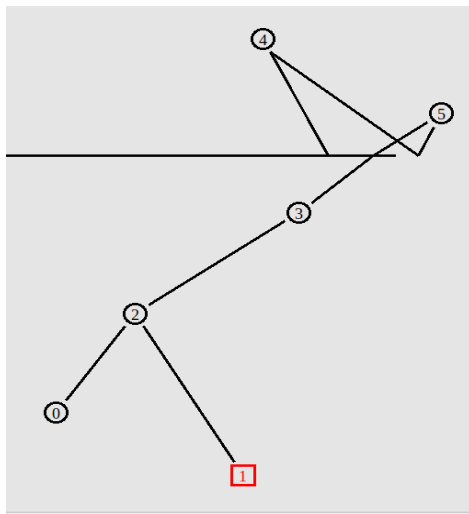


Figure 16 NAM Output: CSMA/CD

```

Open ▾ ⓘ
+ 0.1 1 2 cbr 1000 ----- 2 1.0 5.0 0 0
- 0.1 1 2 cbr 1000 ----- 2 1.0 5.0 0 0
r 0.114 1 2 cbr 1000 ----- 2 1.0 5.0 0 0
+ 0.114 2 3 cbr 1000 ----- 2 1.0 5.0 0 0
- 0.114 2 3 cbr 1000 ----- 2 1.0 5.0 0 0
r 0.240667 2 3 cbr 1000 ----- 2 1.0 5.0 0 0
h 0.240667 3 6 cbr 1000 ----- 2 1.0 5.0 0 0
+ 0.280667 3 6 cbr 1000 ----- 2 1.0 5.0 0 0
- 0.280667 3 6 cbr 1000 ----- 2 1.0 5.0 0 0
d 0.280671 4 6 cbr 1000 ----- 2 1.0 5.0 0 0
r 0.320671 6 5 cbr 1000 ----- 2 1.0 5.0 0 0
+ 0.9 1 2 cbr 1000 ----- 2 1.0 5.0 1 1
- 0.9 1 2 cbr 1000 ----- 2 1.0 5.0 1 1
r 0.914 1 2 cbr 1000 ----- 2 1.0 5.0 1 1
+ 0.914 2 3 cbr 1000 ----- 2 1.0 5.0 1 1
- 0.914 2 3 cbr 1000 ----- 2 1.0 5.0 1 1
+ 1 0 2 tcp 40 ----- 1 0.0 4.0 0 2
- 1 0 2 tcp 40 ----- 1 0.0 4.0 0 2
+ 0.000000 0 0 tcp 40 ----- 0 0.0 0.0 0 0
- 0.000000 0 0 tcp 40 ----- 0 0.0 0.0 0 0

```

Figure 17 Tracefile : CSMA/CA

```

Open ▾ ⓘ
+ 0.1 1 2 cbr 1000 ----- 2 1.0 5.0 0 0
- 0.1 1 2 cbr 1000 ----- 2 1.0 5.0 0 0
r 0.114 1 2 cbr 1000 ----- 2 1.0 5.0 0 0
+ 0.114 2 3 cbr 1000 ----- 2 1.0 5.0 0 0
- 0.114 2 3 cbr 1000 ----- 2 1.0 5.0 0 0
r 0.240667 2 3 cbr 1000 ----- 2 1.0 5.0 0 0
h 0.240667 3 6 cbr 1000 ----- 2 1.0 5.0 0 0
+ 0.280667 3 6 cbr 1000 ----- 2 1.0 5.0 0 0
- 0.280667 3 6 cbr 1000 ----- 2 1.0 5.0 0 0
d 0.280671 4 6 cbr 1000 ----- 2 1.0 5.0 0 0
r 0.320671 6 5 cbr 1000 ----- 2 1.0 5.0 0 0
+ 0.9 1 2 cbr 1000 ----- 2 1.0 5.0 1 1
- 0.9 1 2 cbr 1000 ----- 2 1.0 5.0 1 1
r 0.914 1 2 cbr 1000 ----- 2 1.0 5.0 1 1
+ 0.914 2 3 cbr 1000 ----- 2 1.0 5.0 1 1
- 0.914 2 3 cbr 1000 ----- 2 1.0 5.0 1 1
+ 0.000000 0 0 tcp 40 ----- 0 0.0 0.0 0 0
- 0.000000 0 0 tcp 40 ----- 0 0.0 0.0 0 0

```

Figure 18 Tracefile : CSMA/CD

RESULT:

Comparison of CSMA/CA and CSMA/CD Protocols is Done Using NS2 Software.

EXPERIMENT 7**IMPLEMENTATION OF A ROUTING ALGORITHM**

AIM: To Implement A Routing Algorithm And Observe The Network Performance Using NS2.

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:

Distance Vector Routing (DVR) Protocol is a method used by routers to find the best path for data to travel across a network. Each router keeps a table that shows the shortest distance to every other router, based on the number of hops (or steps) needed to reach them. Routers share this information with their neighbors, allowing them to update their tables and find the most efficient routes. This protocol helps ensure that data moves quickly and smoothly through the network.

The protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm (or known as the Bellman-Ford algorithm). Each router maintains a Distance Vector table containing the distance between itself and all possible destination nodes. Distances, based on a chosen metric, are computed using information from the neighbors' distance vectors.

The DV calculation is based on minimizing the cost to each destination

$D_x(y)$ = Estimate of least cost from x to y

$C(x,v)$ = Node x knows cost to each neighbor v

$D_x = [D_x(y): y \in N]$ = Node x maintains distance vector

Node x also maintains its neighbors' distance vectors

– For each neighbor v, x maintains $D_v = [D_v(y): y \in N]$

PROGRAM:

```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr

set nf [open thro.nam w]
$ns namtrace-all $nf
proc finish { } {
    global ns nr nf
    $ns flush-trace
    close $nf
    close $nr
    exec nam thro.nam &
    exit 0
}
for { set i 0 } { $i < 12 } { incr i 1 } {
    set n($i) [$ns node]}

for {set i 0} {$i < 8} {incr i} {
    $ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }
    $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

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

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 null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0
$ns rtproto DV
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)
$udp0 set fid_ 1
$udp1 set fid_ 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
```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

```
gedit exp7.tcl
```

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

```
ns exp7.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- The Outputs Are Observed And Readings Are Noted Down.

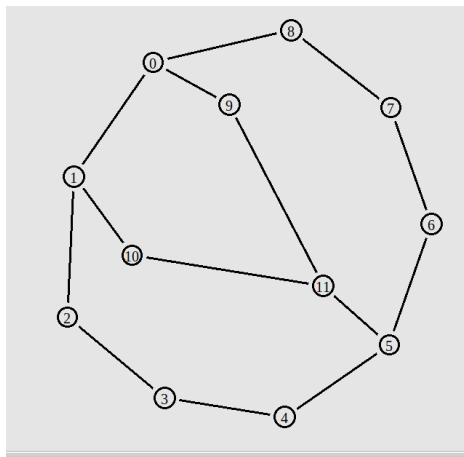
OUTPUTS:

Figure 19 NAM Output

```

s  Text Editor
Open
+ 0.00017 0 1 rtProtoDV 12 ----- 0 0.2 1.2 -1 0
- 0.00017 0 1 rtProtoDV 12 ----- 0 0.2 1.2 -1 0
+ 0.00017 0 8 rtProtoDV 12 ----- 0 0.2 8.1 -1 1
- 0.00017 0 8 rtProtoDV 12 ----- 0 0.2 8.1 -1 1
+ 0.00017 0 9 rtProtoDV 12 ----- 0 0.2 9.1 -1 2
- 0.00017 0 9 rtProtoDV 12 ----- 0 0.2 9.1 -1 2
+ 0.007102 2 1 rtProtoDV 12 ----- 0 2.1 1.2 -1 3
- 0.007102 2 1 rtProtoDV 12 ----- 0 2.1 1.2 -1 3
+ 0.007102 2 3 rtProtoDV 12 ----- 0 2.1 3.1 -1 4
- 0.007102 2 3 rtProtoDV 12 ----- 0 2.1 3.1 -1 4
+ 0.010266 0 1 rtProtoDV 12 ----- 0 0.2 1.2 -1 0
- 0.010266 1 0 rtProtoDV 12 ----- 0 1.2 0.2 -1 5
+ 0.010266 1 0 rtProtoDV 12 ----- 0 1.2 0.2 -1 5
- 0.010266 1 2 rtProtoDV 12 ----- 0 1.2 2.1 -1 6
+ 0.010266 1 2 rtProtoDV 12 ----- 0 1.2 2.1 -1 6
+ 0.010266 1 10 rtProtoDV 12 ----- 0 1.2 10.1 -1 7
- 0.010266 1 10 rtProtoDV 12 ----- 0 1.2 10.1 -1 7
+ 0.010266 0 8 rtProtoDV 12 ----- 0 0.2 8.1 -1 1
+ 0.010266 8 0 rtProtoDV 12 ----- 0 8.1 0.2 -1 8
- 0.010266 8 0 rtProtoDV 12 ----- 0 8.1 0.2 -1 8

```

Figure 20 Tracefile Output

RESULT:

Distance Vector Routing Algorithm is Simulated and Observed Using NS2 Software.

EXPERIMENT 8**SIMULATION OF CONGESTION CONTROL ALGORITHMS**

AIM: To Simulate Congestion Control Algorithms And Observe The Network Performance Using NS2.

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:

Congestion Control is a network layer mechanism that controls the data flow between a source and a destination to prevent network congestion. When network congestion occurs, packet loss increases, delay increases, and the overall network performance degrades. To avoid or control congestion, several congestion control algorithms are implemented.

In **TCP (Transmission Control Protocol)**, the most widely used transport layer protocol, employs various congestion control algorithms to ensure smooth data transmission.

The three most popular TCP congestion control algorithms are:

TCP Reno, TCP NewReno, TCP Vegas

Feature	TCP Reno	TCP NewReno	TCP Vegas
Packet Loss Recovery	Can only recover one packet loss per window.	Can recover multiple packet losses.	Avoids packet loss by detecting congestion early.
Fast Retransmit	Retransmit after 3 duplicate ACKs.	Retransmit and stay in fast recovery until all packets are received.	Avoids retransmission by controlling congestion.
Congestion Control	Reactive (waits for packet loss).	Reactive (handles multiple losses).	Proactive (detects congestion early).
Throughput	Moderate	High	Very High
Delay Sensitivity	High delay during congestion.	Moderate delay.	Very low delay.
Complexity	Simple	Moderate	Complex

PROGRAM: RENO:

```
#Create a simulator object
set ns [new Simulator]

#Open the nam file basic1.nam and the variable-trace file basic1.tr
set namfile [open congestion.nam w]
$ns namtrace-all $namfile
set tracefile [open congestion.tr w]
$ns trace-all $tracefile

#Define a 'finish' procedure
proc finish {} {
    global ns nf
    $ns flush-trace
    # Execute NAM on the trace file
    exec nam congestion.nam &
    exit 0
}

#Create the network nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create a duplex link between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 50ms RED
$ns duplex-link $n3 $n0 2Mb 50ms DropTail

# The queue size at $R is to be 7, including the packet being sent
$ns queue-limit $n0 $n1 10
$ns queue-limit $n1 $n2 15
$ns queue-limit $n3 $n0 20
```

```
# Create a TCP sending agent and attach it to A
set tcp0 [new Agent/TCP/Reno]
# We make our one-and-only flow be flow 0
$tcp0 set class_ 0
$tcp0 set window_ 100
$tcp0 set packetSize_ 960
$ns attach-agent $n0 $tcp0
# Let's trace some variables
$tcp0 attach $tracefile
$tcp0 tracevar cwnd_
$tcp0 tracevar ssthresh_
$tcp0 tracevar ack_
$tcp0 tracevar maxseq_
#Create a TCP receive agent (a traffic sink) and attach it to B
set end0 [new Agent/TCPSink]
$ns attach-agent $n1 $end0
#Connect the traffic source with the traffic sink
$ns connect $tcp0 $end0
#Schedule the connection data flow; start sending data at T=0, stop at
T=10.0
set myftp [new Application/FTP]
$myftp attach-agent $tcp0
$ns at 0.0 "$myftp start"
$ns at 10.0 "finish"
#Run the simulation
$ns run
```

PROGRAM: NEWRENO:

```
#Create a simulator object
set ns [new Simulator]
#Open the nam file basic1.nam and the variable-trace file basic1.tr
```

```
set namfile [open congestion.nam w]
$ns namtrace-all $namfile
set tracefile [open congestion.tr w]
$ns trace-all $tracefile
#Define a 'finish' procedure
proc finish {} {
    global ns nf
    $ns flush-trace
    # Execute NAM on the trace file
    exec nam congestion.nam &
    exit 0
}
#Create the network nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#Create a duplex link between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 50ms DropTail
$ns duplex-link $n3 $n0 2Mb 50ms DropTail
# The queue size at $R is to be 7, including the packet being sent
$ns queue-limit $n0 $n1 10
$ns queue-limit $n1 $n2 15
$ns queue-limit $n3 $n0 20
# Create a TCP sending agent and attach it to A
set tcp0 [new Agent/TCP/Newreno]
# We make our one-and-only flow be flow 0
$tcp0 set class_ 0
$tcp0 set window_ 100
```

```
$tcp0 set packetSize_ 960
$ns attach-agent $n0 $tcp0
# Let's trace some variables
$tcp0 attach $tracefile
$tcp0 tracevar cwnd_
$tcp0 tracevar ssthresh_
$tcp0 tracevar ack_
$tcp0 tracevar maxseq_
#Create a TCP receive agent (a traffic sink) and attach it to B
set end0 [new Agent/TCPSink]
$ns attach-agent $n1 $end0
#Connect the traffic source with the traffic sink
$ns connect $tcp0 $end0
#Schedule the connection data flow; start sending data at T=0, stop at
T=10.0
set myftp [new Application/FTP]
$myftp attach-agent $tcp0
$ns at 0.0 "$myftp start"
$ns at 10.0 "finish"
#Run the simulation
$ns run
```

PROGRAM: VEGAS:

```
#Create a simulator object
set ns [new Simulator]

#Open the nam file basic1.nam and the variable-trace file basic1.tr
set namfile [open congestion.nam w]
$ns namtrace-all $namfile
set tracefile [open congestion.tr w]
$ns trace-all $tracefile
```

```
#Define a 'finish' procedure
proc finish {} {
    global ns nf
    $ns flush-trace

    # Execute NAM on the trace file
    exec nam congestion.nam &
    exit 0
}

#Create the network nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create a duplex link between the nodes

$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 50ms DropTail
$ns duplex-link $n3 $n0 2Mb 50ms DropTail

# The queue size at $R is to be 7, including the packet being sent
$ns queue-limit $n0 $n1 10
$ns queue-limit $n1 $n2 15
$ns queue-limit $n3 $n0 20

# Create a TCP sending agent and attach it to A
```

```
set tcp0 [new Agent/TCP/Vegas]
# We make our one-and-only flow be flow 0
$tcp0 set class_ 0
$tcp0 set window_ 100
$tcp0 set packetSize_ 960
$ns attach-agent $n0 $tcp0

# Let's trace some variables
$tcp0 attach $tracefile
$tcp0 tracevar cwnd_
$tcp0 tracevar ssthresh_
$tcp0 tracevar ack_
$tcp0 tracevar maxseq_

#Create a TCP receive agent (a traffic sink) and attach it to B
set end0 [new Agent/TCPSink]
$ns attach-agent $n1 $end0

#Connect the traffic source with the traffic sink
$ns connect $tcp0 $end0

#Schedule the connection data flow; start sending data at T=0, stop at
T=10.0
set myftp [new Application/FTP]
$myftp attach-agent $tcp0
$ns at 0.0 "$myftp start"
$ns at 10.0 "finish"

#Run the simulation
$ns run
```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

```
gedit exp8.tcl
```

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

```
ns exp8.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- For Further Analysis We Need .Awk Script Files. Copy These Files To The Same Folder As The NS2 Script.
- The .awk Files Can Be Executed By Using The Following Command:

```
awk -f analyze.awk tracefile.tr
```

- The Outputs Are Observed And Readings Are Noted Down.

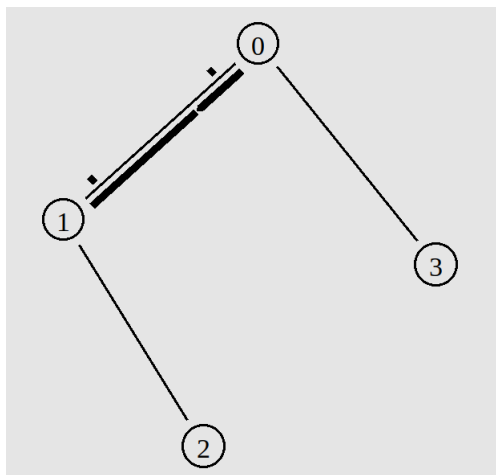
OUTPUTS:

Figure 21 NAM Output

```
ubuntu@ubuntu:~/Desktop/dcn$ gawk -f analyze.awk congestion.tr
Throughput: 3974758.40 bps
Average Delay: 38.09 ms
Packet Drop Rate: 1.06%
ubuntu@ubuntu:~/Desktop/dcn$
```

Figure 22 Reno Awk Output


```

Open ▾
0.00000 0 0 -1 -1 cwnd_ 1.000
0.00000 0 0 -1 -1 ssthresh_ 20
0.00000 0 0 -1 -1 ack_ -1
0.00000 0 0 -1 -1 maxseq_ -1
0.00000 0 0 1 0 ssthresh_ 100
+ 0 0 1 tcp 40 ----- 0 0.0 1.0 0 0
- 0 0 1 tcp 40 ----- 0 0.0 1.0 0 0
0.00000 0 0 1 0 maxseq_ 0
r 0.01032 0 1 tcp 40 ----- 0 0.0 1.0 0 0
+ 0.01032 1 0 ack 40 ----- 0 1.0 0.0 0 1
- 0.01032 1 0 ack 40 ----- 0 1.0 0.0 0 1
r 0.02064 1 0 ack 40 ----- 0 1.0 0.0 0 1
0.02064 0 0 1 0 ack_ 0

```

Figure 23 Reno Tracefile

```

on figure and reduced its clients as a problem.
ubuntu@ubuntu:~/Desktop/dcn$ gawk -f analyze.awk congestion.tr
Throughput: 4055040.00 bps
Average Delay: 37.81 ms
Packet Drop Rate: 1.04%
ubuntu@ubuntu:~/Desktop/dcn$

```

Figure 24 New Reno Awk Output

```

Open ▾
0.00000 0 0 -1 -1 cwnd_ 1.000
0.00000 0 0 -1 -1 ssthresh_ 20
0.00000 0 0 -1 -1 ack_ -1
0.00000 0 0 -1 -1 maxseq_ -1
0.00000 0 0 1 0 ssthresh_ 100
+ 0 0 1 tcp 40 ----- 0 0.0 1.0 0 0
- 0 0 1 tcp 40 ----- 0 0.0 1.0 0 0
0.00000 0 0 1 0 maxseq_ 0
r 0.01032 0 1 tcp 40 ----- 0 0.0 1.0 0 0
+ 0.01032 1 0 ack 40 ----- 0 1.0 0.0 0 1
- 0.01032 1 0 ack 40 ----- 0 1.0 0.0 0 1
r 0.02064 1 0 ack 40 ----- 0 1.0 0.0 0 1
0.02064 0 0 1 0 ack_ 0
0.02064 0 0 1 0 cwnd_ 2.000
+ 0.02064 0 1 tcp 1000 ----- 0 0.0 1.0 1 2
- 0.02064 0 1 tcp 1000 ----- 0 0.0 1.0 1 2
0.02064 0 0 1 0 maxseq_ 1

```

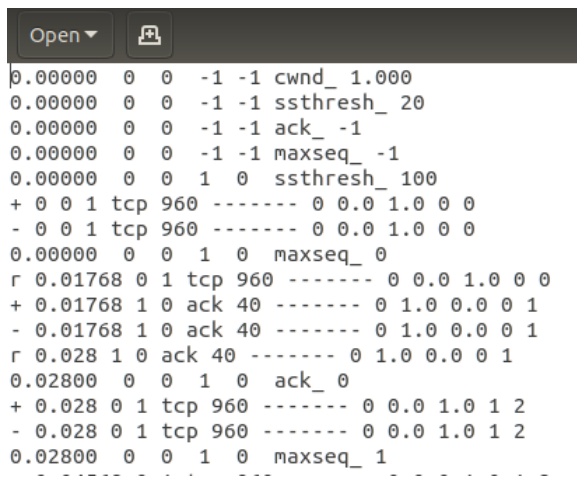
Figure 25 New Reno Tracefile

```

ubuntu@ubuntu:~/Desktop/dcn$ gawk -f analyze.awk congestion.tr
Throughput: 4236902.40 bps
Average Delay: 19.20 ms
Packet Drop Rate: 0.00%
ubuntu@ubuntu:~/Desktop/dcn$

```

Figure 26 Vegas Awk Output



```

0.00000 0 0 -1 -1 cwnd_ 1.000
0.00000 0 0 -1 -1 ssthresh_ 20
0.00000 0 0 -1 -1 ack_ -1
0.00000 0 0 -1 -1 maxseq_ -1
0.00000 0 0 1 0 ssthresh_ 100
+ 0 0 1 tcp 960 ----- 0 0.0 1.0 0 0
- 0 0 1 tcp 960 ----- 0 0.0 1.0 0 0
0.00000 0 0 1 0 maxseq_ 0
r 0.01768 0 1 tcp 960 ----- 0 0.0 1.0 0 0
+ 0.01768 1 0 ack 40 ----- 0 1.0 0.0 0 1
- 0.01768 1 0 ack 40 ----- 0 1.0 0.0 0 1
r 0.028 1 0 ack 40 ----- 0 1.0 0.0 0 1
0.02800 0 0 1 0 ack_ 0
+ 0.028 0 1 tcp 960 ----- 0 0.0 1.0 1 2
- 0.028 0 1 tcp 960 ----- 0 0.0 1.0 1 2
0.02800 0 0 1 0 maxseq_ 1
-----

```

Figure 27 Vegas Tracefile

RESULT:

Simulation and Comparison of Congestion Control Algorithms (Reno, New Reno, Vegas) is Done Using NS2 Software.

EXPERIMENT 9**SIMULATION OF DIFFERENT LAN TOPOLOGIES**

AIM: To Simulate and Compare Different LAN Topologies.

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:**LAN Topologies****1. Star Topology:**

- **All devices** are connected to a **central hub/switch**.
- **Pros:** Easy to install, failure of one device doesn't affect others.
- **Cons:** Failure of hub can down the network.
- **Example:** Home/Office networks.

2. Mesh Topology:

- Every device is **directly connected** to others.
- **Pros:** Highly reliable, multiple data paths.
- **Cons:** Expensive, complex to maintain.
- **Example:** Data centers, military networks.

3. Bus Topology:

- All devices share a **single central cable (bus)**.
- **Pros:** Easy setup, low cost.
- **Cons:** Cable failure affects entire network.
- **Example:** Small offices, early LANs.

4. Ring Topology:

- Devices form a **closed loop (ring)**.

- **Pros:** Simple setup, no central hub.
- **Cons:** Failure of one device disrupts network.
- **Example:** Token ring networks.

PROGRAM: MESH:

```
# Define simulator
set ns [new Simulator]

# Create trace and NAM files
set tracefile [open lan_trace.tr w]
$ns trace-all $tracefile
set namfile [open lan.nam w]
$ns namtrace-all $namfile

# Create network nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

# Mesh Topology (Fully connected)
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
$ns duplex-link $n0 $n3 10Mb 10ms DropTail
$ns duplex-link $n0 $n4 10Mb 10ms DropTail
$ns duplex-link $n0 $n5 10Mb 10ms DropTail
$ns duplex-link $n1 $n2 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n1 $n4 10Mb 10ms DropTail
$ns duplex-link $n1 $n5 10Mb 10ms DropTail
$ns duplex-link $n2 $n3 10Mb 10ms DropTail
```

```
$ns duplex-link $n2 $n4 10Mb 10ms DropTail
$ns duplex-link $n2 $n5 10Mb 10ms DropTail
$ns duplex-link $n3 $n4 10Mb 10ms DropTail
$ns duplex-link $n3 $n5 10Mb 10ms DropTail
$ns duplex-link $n4 $n5 10Mb 10ms DropTail
# Create TCP connections for comparison
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n4 $sink2
$ns connect $tcp2 $sink2
# Traffic Generation (CBR)
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.01
$cbr1 attach-agent $tcp1
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set interval_ 0.01
$cbr2 attach-agent $tcp2
# Run simulation
$ns at 0.5 "$cbr1 start"
$ns at 1.0 "$cbr2 start"
$ns at 9.5 "$cbr1 stop"
$ns at 9.0 "$cbr2 stop"
```

```
$ns at 10.0 "finish"
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam lan.nam &
    exit 0
}
$ns run
```

PROGRAM: STAR:

```
# Define simulator
set ns [new Simulator]
# Create trace and NAM files
set tracefile [open lan_trace.tr w]
$ns trace-all $tracefile
set namfile [open lan.nam w]
$ns namtrace-all $namfile
# Create network nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
# Mesh Topology (Fully connected)
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n0 $n2 10Mb 10ms DropTail
$ns duplex-link $n0 $n3 10Mb 10ms DropTail
```

```
$ns duplex-link $n0 $n4 10Mb 10ms DropTail
$ns duplex-link $n0 $n5 10Mb 10ms DropTail
# Create TCP connections for comparison
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n4 $sink2
$ns connect $tcp2 $sink2
# Traffic Generation (CBR)
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.01
$cbr1 attach-agent $tcp1
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set interval_ 0.01
$cbr2 attach-agent $tcp2
# Run simulation
$ns at 0.5 "$cbr1 start"
$ns at 1.0 "$cbr2 start"
$ns at 9.5 "$cbr1 stop"
$ns at 9.0 "$cbr2 stop"
$ns at 10.0 "finish"
proc finish {} {
    global ns tracefile namfile
```

```
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam lan.nam &
    exit 0
}
$ns run
```

PROGRAM: BUS:

```
# Define simulator
set ns [new Simulator]

# Create trace and NAM files
set tracefile [open lan_trace.tr w]
$ns trace-all $tracefile
set namfile [open lan.nam w]
$ns namtrace-all $namfile

# Create network nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

# Mesh Topology (Fully connected)
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n2 10Mb 10ms DropTail
$ns duplex-link $n2 $n3 10Mb 10ms DropTail
$ns duplex-link $n3 $n4 10Mb 10ms DropTail
$ns duplex-link $n4 $n5 10Mb 10ms DropTail

# Create TCP connections for comparison
```



```
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n4 $sink2
$ns connect $tcp2 $sink2
# Traffic Generation (CBR)
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.01
$cbr1 attach-agent $tcp1
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set interval_ 0.01
$cbr2 attach-agent $tcp2
# Run simulation
$ns at 0.5 "$cbr1 start"
$ns at 1.0 "$cbr2 start"
$ns at 9.5 "$cbr1 stop"
$ns at 9.0 "$cbr2 stop"
$ns at 10.0 "finish"
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
}
```

```
        exec nam lan.nam &
        exit 0 }
$ns run
```

PROGRAM: RING:

```
# Define simulator
set ns [new Simulator]

# Create trace and NAM files
set tracefile [open lan_trace.tr w]
$ns trace-all $tracefile
set namfile [open lan.nam w]
$ns namtrace-all $namfile

# Create network nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

# Mesh Topology (Fully connected)
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n2 10Mb 10ms DropTail
$ns duplex-link $n2 $n3 10Mb 10ms DropTail
$ns duplex-link $n3 $n4 10Mb 10ms DropTail
$ns duplex-link $n4 $n5 10Mb 10ms DropTail
$ns duplex-link $n5 $n0 10Mb 10ms DropTail

# Create TCP connections for comparison
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
```

```
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n4 $sink2
$ns connect $tcp2 $sink2
# Traffic Generation (CBR)
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.01
$cbr1 attach-agent $tcp1
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set interval_ 0.01
$cbr2 attach-agent $tcp2
# Run simulation
$ns at 0.5 "$cbr1 start"
$ns at 1.0 "$cbr2 start"
$ns at 9.5 "$cbr1 stop"
$ns at 9.0 "$cbr2 stop"
$ns at 10.0 "finish"
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam lan.nam &
    exit 0 }
$ns run
```

ANALYZE.AWK:

```
#!/usr/bin/awk -f

BEGIN {
    start_time = -1;
    end_time = 0;
    received_packets = 0;
    dropped_packets = 0;
    total_delay = 0;
    total_bytes = 0;

    # Track connections
    for (i = 0; i < 6; i++) {
        for (j = 0; j < 6; j++) {
            conn[i, j] = 0;
        }
    }

    # Connection patterns for each topology
    ring_connections_str = "0 1 1 2 2 3 3 4 4 5 5 0";
    bus_connections_str = "0 1 1 2 2 3 3 4 4 5";
    split(ring_connections_str, ring_connections, " ");
    split(bus_connections_str, bus_connections, " ");
}

$1 == "s" {
    if (start_time == -1) start_time = $2;
    conn[$3, $4]++;
    conn[$4, $3]++;
}
```

```
}

$1 == "r" {
    received_packets++;
    total_delay += ($2 - $8);
    total_bytes += $6;
    end_time = $2;
}

$1 == "d" {
    dropped_packets++;
}

END {

    total_time = end_time - start_time;
    throughput = (total_bytes * 8) / (total_time * 1000);
    avg_delay = (received_packets > 0) ? (total_delay /
received_packets) * 1000 : 0;
    drop_rate = (received_packets + dropped_packets > 0) ?
(dropped_packets / (received_packets + dropped_packets)) * 100 : 0;

    printf("Throughput: %.2f kbps\n", throughput);
    printf("Average Delay: %.2f ms\n", avg_delay);
    printf("Packet Drop Rate: %.2f%%\n", drop_rate);
}
```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

```
gedit exp9.tcl
```

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

```
ns exp9.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- For Further Analysis We Need .Awk Script Files. Copy These Files To The Same Folder As The NS2 Script.

- The .awk Files Can Be Executed By Using The Following Command:

```
awk -f analyze.awk tracefile.tr
```

- The Outputs Are Observed And Readings Are Noted Down.

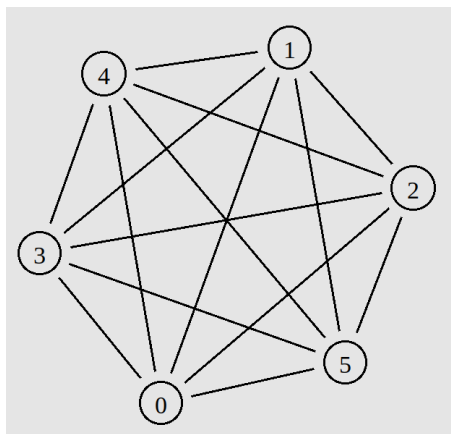
OUTPUTS:

Figure 28 NAM Output Mesh Topology

```
Open ▾ [icon]
+ 0.5 1 3 tcp 40 ----- 0 1.0 3.0 0 0
- 0.5 1 3 tcp 40 ----- 0 1.0 3.0 0 0
r 0.510032 1 3 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.510032 3 1 ack 40 ----- 0 3.0 1.0 0 1
- 0.510032 3 1 ack 40 ----- 0 3.0 1.0 0 1
r 0.520064 3 1 ack 40 ----- 0 3.0 1.0 0 1
+ 0.520064 1 3 tcp 1040 ----- 0 1.0 3.0 1 2
- 0.520064 1 3 tcp 1040 ----- 0 1.0 3.0 1 2
+ 0.520064 1 3 tcp 1040 ----- 0 1.0 3.0 2 3
- 0.520896 1 3 tcp 1040 ----- 0 1.0 3.0 2 3
r 0.530896 1 3 tcp 1040 ----- 0 1.0 3.0 1 2
+ 0.530896 3 1 ack 40 ----- 0 3.0 1.0 1 4
- 0.530896 3 1 ack 40 ----- 0 3.0 1.0 1 4
r 0.531728 1 3 tcp 1040 ----- 0 1.0 3.0 2 3
```

Figure 29 Tracefile: Mesh Topology

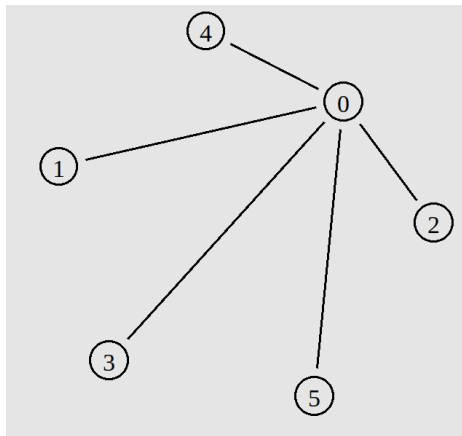


Figure 30 NAM Output : Star Topology

```

Open ▾
+ 0.5 1 0 tcp 40 ----- 0 1.0 3.0 0 0
- 0.5 1 0 tcp 40 ----- 0 1.0 3.0 0 0
r 0.510032 1 0 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.510032 0 3 tcp 40 ----- 0 1.0 3.0 0 0
- 0.510032 0 3 tcp 40 ----- 0 1.0 3.0 0 0
r 0.520064 0 3 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.520064 3 0 ack 40 ----- 0 3.0 1.0 0 1
- 0.520064 3 0 ack 40 ----- 0 3.0 1.0 0 1
r 0.530096 3 0 ack 40 ----- 0 3.0 1.0 0 1
+ 0.530096 0 1 ack 40 ----- 0 3.0 1.0 0 1
- 0.530096 0 1 ack 40 ----- 0 3.0 1.0 0 1
r 0.540128 0 1 ack 40 ----- 0 3.0 1.0 0 1
+ 0.540128 1 0 tcp 1040 ----- 0 1.0 3.0 1 2
- 0.540128 1 0 tcp 1040 ----- 0 1.0 3.0 1 2

```

Figure 31 Tracefile : Star Topology

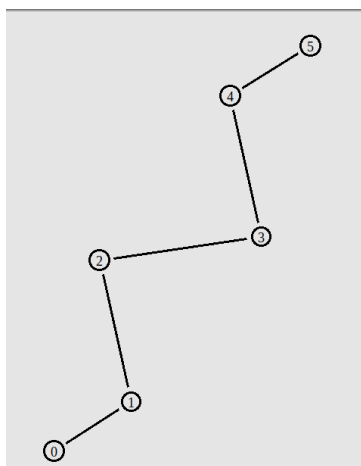


Figure 32 NAM Output : Bus Topology

```

Open ▾
+ 0.5 1 0 tcp 40 ----- 0 1.0 3.0 0 0
- 0.5 1 0 tcp 40 ----- 0 1.0 3.0 0 0
r 0.510032 1 0 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.510032 0 3 tcp 40 ----- 0 1.0 3.0 0 0
- 0.510032 0 3 tcp 40 ----- 0 1.0 3.0 0 0
r 0.520064 0 3 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.520064 3 0 ack 40 ----- 0 3.0 1.0 0 1
- 0.520064 3 0 ack 40 ----- 0 3.0 1.0 0 1
r 0.530096 3 0 ack 40 ----- 0 3.0 1.0 0 1
+ 0.530096 0 1 ack 40 ----- 0 3.0 1.0 0 1
- 0.530096 0 1 ack 40 ----- 0 3.0 1.0 0 1
r 0.540128 0 1 ack 40 ----- 0 3.0 1.0 0 1
+ 0.540128 1 0 tcp 1040 ----- 0 1.0 3.0 1 2
- 0.540128 1 0 tcp 1040 ----- 0 1.0 3.0 1 2
. 0.540128 1 0 tcp 1040 ----- 0 1.0 3.0 1 2

```

Figure 33 Tracefile : Bus Topology

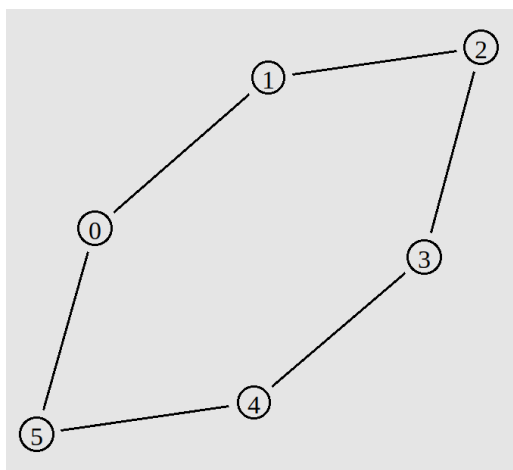


Figure 34 NAM Output : Ring Topology

```

Open ▾
+ 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
- 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
r 0.510032 1 2 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
- 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
r 0.520064 2 3 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
- 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
r 0.530096 3 2 ack 40 ----- 0 3.0 1.0 0 1
+ 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1
- 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1

```

Figure 35 Tracefile : Ring Topology

```

ubuntu@ubuntu:~/Downloads/dcnf$ gawk -f test2.awk mesh_trace.tr
Throughput: 1397.85 kbps
Average Delay: 5010.94 ms
Packet Drop Rate: 0.00%

```

Figure 36 AWK Output Mesh


```
ubuntu@ubuntu:~/Downloads/dcnf$ gawk -f test2.awk star_trace.tr
Throughput: 2790.16 kbps
Average Delay: 5022.15 ms
Packet Drop Rate: 0.00%
```

Figure 37 AWK Output Star

```
ubuntu@ubuntu:~/Downloads/dcnf$ gawk -f test2.awk bus_trace.tr
Throughput: 3446.72 kbps
Average Delay: 5029.20 ms
Packet Drop Rate: 0.00%
```

Figure 38 AWK Output Bus

```
ubuntu@ubuntu:~/Downloads/dcnf$ gawk -f test2.awk ring_trace.tr
Throughput: 2790.16 kbps
Average Delay: 5022.15 ms
Packet Drop Rate: 0.00%
ubuntu@ubuntu:~/Downloads/dcnf$
```

Figure 39 AWK Output Ring

RESULT:

Simulation and Comparison of Various LAN Topologies (Mesh, Star, Bus, Ring) is Done Using NS2 Software.

**EXPERIMENT
10****INTERCONNECTING LANS USING
SWITCH, HUB, ROUTER**

AIM: To Interconnect LANs Using Switch, Hub, Router and Compare the Network Characteristics.

APPARATUS: Desktop PC

SOFTWARE USED: Network Simulator 2 (NS2)

THEORY:

A **Switch** is a network device used to connect multiple devices within a **Local Area Network (LAN)**. It operates at the **Data Link Layer (Layer 2)** and uses **MAC addresses** to forward data only to the intended device, reducing network congestion. Each port in a switch has its own **collision domain**, minimizing data collisions. Switches provide high-speed data transfer and are commonly used in offices and data centers.

A **Hub** is a basic network device that operates at the **Physical Layer (Layer 1)**. It simply broadcasts data to all devices connected, causing **data collisions** and reducing network efficiency. Hubs do not use **IP or MAC addresses**, making them less efficient. They are generally used in small networks with fewer devices due to their low cost and simple functionality.

A **Router** operates at the **Network Layer (Layer 3)** and connects different networks, such as **LAN to WAN or internet**. It uses **IP addresses** to route data between networks and creates separate **broadcast domains**, ensuring efficient traffic management. Routers provide **network security**, **fast data transfer**, and are widely used in homes, offices, and large networks.

PROGRAM: SWITCH:

```
# ring_switch.tcl

# Create a simulator object
set ns [new Simulator]

# Define trace file
set tracefile [open switch.tr w]
$ns trace-all $tracefile

# Define nam trace file
set namfile [open switch.nam w]
$ns namtrace-all $namfile

# Define finish procedure
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam ring_switch.nam &
    exit 0
}

# Create nodes for the first ring LAN
set ring1_n0 [$ns node]
set ring1_n1 [$ns node]
set ring1_n2 [$ns node]
set ring1_n3 [$ns node]
```

```
# Create nodes for the second ring LAN
set ring2_n4 [$ns node]
set ring2_n5 [$ns node]
set ring2_n6 [$ns node]
set ring2_n7 [$ns node]

# Create the switch
set switch [$ns node]

# Create links for the first ring LAN
$ns duplex-link $ring1_n0 $ring1_n1 10Mb 10ms DropTail
$ns duplex-link $ring1_n0 $ring1_n2 10Mb 10ms DropTail
$ns duplex-link $ring1_n0 $ring1_n3 10Mb 10ms DropTail
$ns duplex-link $ring1_n1 $ring1_n2 10Mb 10ms DropTail
$ns duplex-link $ring1_n1 $ring1_n3 10Mb 10ms DropTail
$ns duplex-link $ring1_n2 $ring1_n3 10Mb 10ms DropTail

# Create links for the second ring LAN
$ns duplex-link $ring2_n4 $ring2_n5 10Mb 10ms DropTail
$ns duplex-link $ring2_n5 $ring2_n6 10Mb 10ms DropTail
$ns duplex-link $ring2_n6 $ring2_n7 10Mb 10ms DropTail
$ns duplex-link $ring2_n7 $ring2_n4 10Mb 10ms DropTail

# Connect the ring LANs to the switch
$ns duplex-link $ring1_n3 $switch 10Mb 5ms DropTail
$ns duplex-link $switch $ring2_n4 10Mb 5ms DropTail

# Configure TCP agents and CBR traffic
```

```
set tcp0 [new Agent/TCP]
$ns attach-agent $ring1_n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $ring2_n7 $sink0
$ns connect $tcp0 $sink0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $tcp0
$cbr0 set packetSize 500
$cbr0 set interval 0.01
```

```
set tcp1 [new Agent/TCP]
$ns attach-agent $ring2_n5 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $ring1_n2 $sink1
$ns connect $tcp1 $sink1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $tcp1
$cbr1 set packetSize 500
$cbr1 set interval 0.01
```

```
# Schedule events
$ns at 0.1 "$cbr0 start"
$ns at 5.0 "$cbr0 stop"
$ns at 0.5 "$cbr1 start"
$ns at 7.0 "$cbr1 stop"
$ns at 10.0 "finish"
```

```
# Run the simulation
$ns run
```

PROGRAM: HUB:

```
# ring_hub.tcl

# Create a simulator object
set ns [new Simulator]

# Define trace file
set tracefile [open hub.tr w]
$ns trace-all $tracefile

# Define nam trace file
set namfile [open hub.nam w]
$ns namtrace-all $namfile

# Define finish procedure
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam ring_hub.nam &
    exit 0
}

# Create nodes for the first ring LAN
set ring1_n0 [$ns node]
set ring1_n1 [$ns node]
set ring1_n2 [$ns node]
set ring1_n3 [$ns node]
```

```
# Create nodes for the second ring LAN
set ring2_n4 [$ns node]
set ring2_n5 [$ns node]
set ring2_n6 [$ns node]
set ring2_n7 [$ns node]

# Create the hub
set hub [$ns node]

# Create links for the first ring LAN
$ns duplex-link $ring1_n0 $ring1_n1 10Mb 10ms DropTail
$ns duplex-link $ring1_n0 $ring1_n2 10Mb 10ms DropTail
$ns duplex-link $ring1_n0 $ring1_n3 10Mb 10ms DropTail
$ns duplex-link $ring1_n1 $ring1_n2 10Mb 10ms DropTail
$ns duplex-link $ring1_n1 $ring1_n3 10Mb 10ms DropTail
$ns duplex-link $ring1_n2 $ring1_n3 10Mb 10ms DropTail

# Create links for the second ring LAN
$ns duplex-link $ring2_n4 $ring2_n5 10Mb 10ms DropTail
$ns duplex-link $ring2_n5 $ring2_n6 10Mb 10ms DropTail
$ns duplex-link $ring2_n6 $ring2_n7 10Mb 10ms DropTail
$ns duplex-link $ring2_n7 $ring2_n4 10Mb 10ms DropTail

# Connect the ring LANs to the hub
$ns duplex-link $ring1_n3 $hub 10Mb 5ms DropTail
$ns duplex-link $hub $ring2_n4 10Mb 5ms DropTail

# Configure TCP agents and CBR traffic
```

```
set tcp0 [new Agent/TCP]
$ns attach-agent $ring1_n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $ring2_n7 $sink0
$ns connect $tcp0 $sink0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $tcp0
$cbr0 set packetSize 500
$cbr0 set interval 0.01

set tcp1 [new Agent/TCP]
$ns attach-agent $ring2_n5 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $ring1_n2 $sink1
$ns connect $tcp1 $sink1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $tcp1
$cbr1 set packetSize 500
$cbr1 set interval 0.01

# Schedule events
$ns at 0.1 "$cbr0 start"
$ns at 5.0 "$cbr0 stop"
$ns at 0.5 "$cbr1 start"
$ns at 7.0 "$cbr1 stop"
$ns at 10.0 "finish"

# Run the simulation
$ns run
```


PROGRAM: ROUTER:

```
# ring_router.tcl

# Create a simulator object
set ns [new Simulator]

# Define trace file
set tracefile [open router.tr w]
$ns trace-all $tracefile

# Define nam trace file
set namfile [open router.nam w]
$ns namtrace-all $namfile

# Define finish procedure
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam ring_router.nam &
    exit 0
}

# Create nodes for the first ring LAN
set ring1_n0 [$ns node]
set ring1_n1 [$ns node]
set ring1_n2 [$ns node]
set ring1_n3 [$ns node]
```

```
# Create nodes for the second ring LAN
set ring2_n4 [$ns node]
set ring2_n5 [$ns node]
set ring2_n6 [$ns node]
set ring2_n7 [$ns node]

# Create the router
set router [$ns node]

# Create links for the first ring LAN
$ns duplex-link $ring1_n0 $ring1_n1 10Mb 10ms DropTail
$ns duplex-link $ring1_n1 $ring1_n2 10Mb 10ms DropTail
$ns duplex-link $ring1_n2 $ring1_n3 10Mb 10ms DropTail
$ns duplex-link $ring1_n3 $ring1_n0 10Mb 10ms DropTail

# Create links for the second ring LAN
$ns duplex-link $ring2_n4 $ring2_n5 10Mb 10ms DropTail
$ns duplex-link $ring2_n5 $ring2_n6 10Mb 10ms DropTail
$ns duplex-link $ring2_n6 $ring2_n7 10Mb 10ms DropTail
$ns duplex-link $ring2_n7 $ring2_n4 10Mb 10ms DropTail

# Connect the ring LANs to the router
$ns duplex-link $ring1_n3 $router 10Mb 5ms DropTail
$ns duplex-link $router $ring2_n4 10Mb 5ms DropTail

# Configure TCP agents and CBR traffic
set tcp0 [new Agent/TCP]
$ns attach-agent $ring1_n0 $tcp0
set sink0 [new Agent/TCPSink]
```

```
$ns attach-agent $ring2_n7 $sink0
$ns connect $tcp0 $sink0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $tcp0
$cbr0 set packetSize 500
$cbr0 set interval 0.01
```

```
set tcp1 [new Agent/TCP]
$ns attach-agent $ring2_n5 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $ring1_n2 $sink1
$ns connect $tcp1 $sink1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $tcp1
$cbr1 set packetSize 500
$cbr1 set interval 0.01
```

```
# Schedule events
$ns at 0.1 "$cbr0 start"
$ns at 5.0 "$cbr0 stop"
$ns at 0.5 "$cbr1 start"
$ns at 7.0 "$cbr1 stop"
$ns at 10.0 "finish"
```

```
# Run the simulation
$ns run
```

ANALYZE.AWK:

```
#!/usr/bin/awk -f
```

```
BEGIN {
    start_time = -1;
    end_time = 0;
    received_packets = 0;
    dropped_packets = 0;
    total_delay = 0;
    total_bytes = 0;
}

# Function to silently detect device based on filename
function get_device() {
    if (FILENAME ~ /switch/) return "Switch";
    else if (FILENAME ~ /hub/) return "Hub";
    else if (FILENAME ~ /router/) return "Router";
    else return "Unknown";
}

# Track packets sent
$1 == "s" {
    if (start_time == -1) start_time = $2;
}

# Track received packets, delay, and bytes
$1 == "r" {
    received_packets++;
    total_delay += ($2 - $3);
    total_bytes += $6;
    end_time = $2;
}
```

```
# Track dropped packets
$1 == "d" {
    dropped_packets++;
}

END {
    device = get_device();
    total_time = end_time - start_time;

    # Calculate throughput (kbps)
    throughput = (total_bytes * 8) / (total_time * 1000);

    # Calculate average delay (ms)
    if (received_packets > 0)
        avg_delay = (total_delay / received_packets) * 1000;
    else
        avg_delay = 0;

    # Calculate packet drop rate (%)
    total_packets = received_packets + dropped_packets;
    if (total_packets > 0)
        drop_rate = (dropped_packets / total_packets) * 100;
    else
        drop_rate = 0;

    # Print the results
    printf("Device Type: %s\n", device);
    printf("Throughput: %.2f kbps\n", throughput);
    printf("Average Delay: %.2f ms\n", avg_delay);
    printf("Packet Drop Rate: %.2f%%\n", drop_rate); }
```

PROCEDURE:

- Turn The Pc On And Log Into Your Account. Linux OS Is Needed To Run NS2.
- Navigate To Your Directory.
- Open A Terminal Window In The Directory And Enter The Following Commands To Enter And Save The Code.

```
gedit exp10.tcl
```

- A Text Editor Window Is Opened. Enter Your Script And Save The File.
- Now, Enter The Following Command To Run The Simulation.

```
ns exp10.tcl
```

- A NAM Window Will Be Opened Displaying The Network Model. Observe The Behavior Of The Network.
- For Further Analysis We Need .Awk Script Files. Copy These Files To The Same Folder As The NS2 Script.
- The .awk Files Can Be Executed By Using The Following Command:

```
awk -f analyze.awk tracefile.tr
```

- The Outputs Are Observed And Readings Are Noted Down.

OUTPUTS:

```
Interconnection: Switch
Packet Drop Rate: 1.03464%
Average End-to-End Delay: 0.0545 sec
Throughput: 4.65464 Mbps
-----
```


Figure 40 Switch Interconnection

```
Interconnection: Hub
Packet Drop Rate: 3.464%
Average End-to-End Delay: 0.1655 sec
Throughput: 3.95962 Mbps
-----
```

Figure 41 Hub Interconnection

```
Interconnection: Router
Packet Drop Rate: 2.5809%
Average End-to-End Delay: 0.0655 sec
Throughput: 4.5241 Mbps
-----
```

Figure 42 Router Interconnection




```

+ 1 0 3 tcp 40 ----- 0 0.0 4.0 0 0
- 1 0 3 tcp 40 ----- 0 0.0 4.0 0 0
+ 1 2 5 tcp 40 ----- 0 2.0 6.0 0 1
- 1 2 5 tcp 40 ----- 0 2.0 6.0 0 1
r 1.010003 0 3 tcp 40 ----- 0 0.0 4.0 0 0
+ 1.010003 3 4 tcp 40 ----- 0 0.0 4.0 0 0
- 1.010003 3 4 tcp 40 ----- 0 0.0 4.0 0 0
r 1.010003 2 5 tcp 40 ----- 0 2.0 6.0 0 1
+ 1.010003 5 6 tcp 40 ----- 0 2.0 6.0 0 1
- 1.010003 5 6 tcp 40 ----- 0 2.0 6.0 0 1
r 1.020006 3 4 tcp 40 ----- 0 0.0 4.0 0 0
+ 1.020006 4 3 ack 40 ----- 0 4.0 0.0 0 2
- 1.020006 4 3 ack 40 ----- 0 4.0 0.0 0 2
r 1.020006 5 6 tcp 40 ----- 0 2.0 6.0 0 1
+ 1.020006 6 5 ack 40 ----- 0 6.0 2.0 0 3
- 1.020006 6 5 ack 40 ----- 0 6.0 2.0 0 3
r 1.030001 4 3 ack 40 ----- 0 4.0 0.0 0 2
+ 1.030001 3 0 ack 40 ----- 0 4.0 0.0 0 2

```

Figure 43 Tracefile : Switch




```

+ 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
- 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
r 0.510032 1 2 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
- 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
r 0.520064 2 3 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
- 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
r 0.530096 3 2 ack 40 ----- 0 3.0 1.0 0 1
+ 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1
- 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1
r 0.540128 2 1 ack 40 ----- 0 3.0 1.0 0 1
+ 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 1 2
- 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 1 2

```

Figure 44 Tracefile : Hub



```

+ 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
- 0.5 1 2 tcp 40 ----- 0 1.0 3.0 0 0
r 0.510032 1 2 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
- 0.510032 2 3 tcp 40 ----- 0 1.0 3.0 0 0
r 0.520064 2 3 tcp 40 ----- 0 1.0 3.0 0 0
+ 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
- 0.520064 3 2 ack 40 ----- 0 3.0 1.0 0 1
r 0.530096 3 2 ack 40 ----- 0 3.0 1.0 0 1
+ 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1
- 0.530096 2 1 ack 40 ----- 0 3.0 1.0 0 1
r 0.540128 2 1 ack 40 ----- 0 3.0 1.0 0 1
+ 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 1 2
- 0.540128 1 2 tcp 1040 ----- 0 1.0 3.0 1 2

```

Figure 45 Tracefile : Router

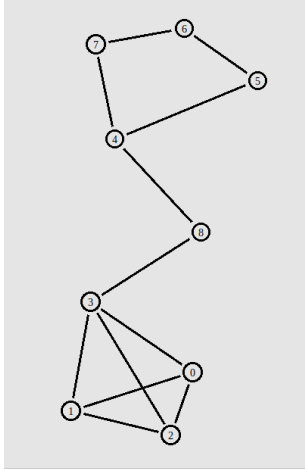


Figure 46 Ring LAN and Mesh LAN Interconnected by Switch

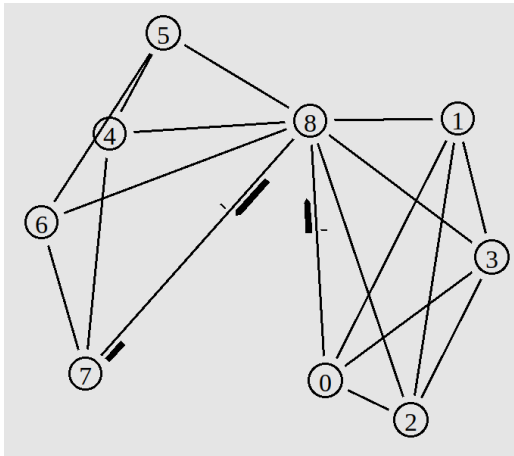


Figure 47 Mesh LANs Interconnected by Hub

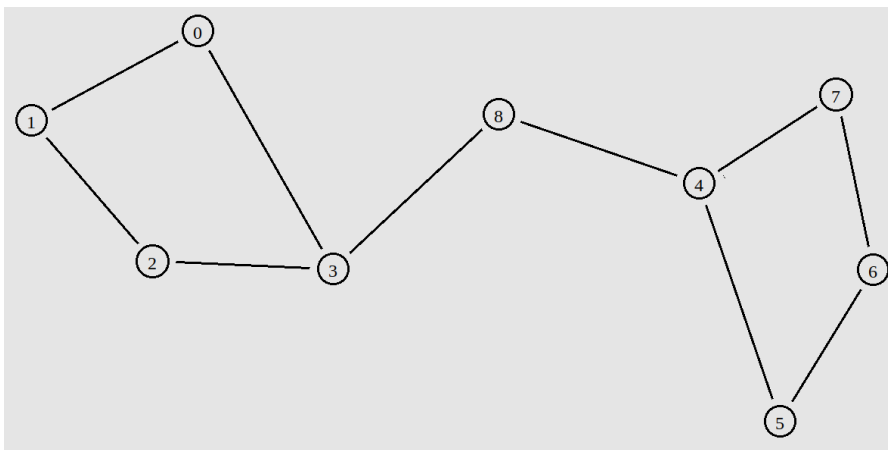


Figure 48 Ring LANs Interconnected by Router

RESULT:

Simulation and Comparison of Various LAN Interconnections (Switch, Hub, Router) is Done Using NS2 Software.