

Course outcomes: On the completion of this laboratory course, the students will be able to:

1. Design and Simulate Network elements with various protocols and standards.
2. Use the network simulator tools for learning and practice of networking algorithms.
3. Demonstrate the working of various protocols and algorithms using C programming.

Programme Outcomes:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: The problems that cannot be solved by straightforward application of knowledge, theories and techniques applicable to the engineering discipline that may not have a unique solution.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modelling to complex engineering activities, with an understanding of the limitations.
6. 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.
7. 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.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. 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.

VTU SYLLABUS

COMPUTER NETWORKS LABORATORY B.E., VI Semester, Electronics & Communication Engineering [As per Choice Based Credit System (CBCS) scheme]

Subject Code -15ECL68

IA Marks -20

Number of Lecture Hours/Week = 03 Hours

Exam Marks 80

Exam Hours 03

Laboratory Experiments

PART-A: Simulation experiments using NS2/ NS3/ OPNET/ NCTUNS/ NetSim/ QualNet/ Packet Tracer or any other equivalent tool

1. Implement a point to point network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth.
2. Implement a four node point to point network with links n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP.
3. Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.
4. Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.
5. Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.
6. Implementation of Link state routing algorithm.

PART-B: Implement the following in C/C++

1. Write a program for a HDLC frame to perform the following.
 - i) Bit stuffing
 - ii) Character stuffing.
2. Write a program for distance vector algorithm to find suitable path for transmission.

3. Implement Dijkstra's algorithm to compute the shortest routing path.
4. For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases
 - a. Without error
 - b. With error
5. Implementation of Stop and Wait Protocol and Sliding Window Protocol
6. Write a program for congestion control using leaky bucket algorithm.

INDEX

Experiments		
Sl No	PART-A	Pg No
1	Implement a point to point network with four nodes and duplex links between them. Analyse the network performance by setting the queue size and varying the bandwidth.	8
2	Implement a four node point to point network with links n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP.	12
3	Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.	17
4	Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.	20
5	Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.	23
6	Implementation of Link state routing algorithm.	26

PART-B

1	Write a program for a HDLC frame to perform the following. i) Bit stuffing. ii) Character stuffing.	31
2	Write a program for distance vector algorithm to find suitable path for transmission.	35
3	Implement Dijkstra's algorithm to compute the shortest routing path.	39
4	For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases. a. Without error. b. With error.	43
5	Implementation of Stop and Wait Protocol and Sliding Window Protocol.	46
6	Write a program for congestion control using leaky bucket algorithm.	51
	Viva Questions	54

PART-A

EXPERIMENTS USING

NETWORK

SIMULATOR 2

EXPERIMENT NO 1

Aim: Implement a point to point network with four nodes and duplex links between them. Analyze the network performance by setting the queue size and varying the bandwidth.

Topology:

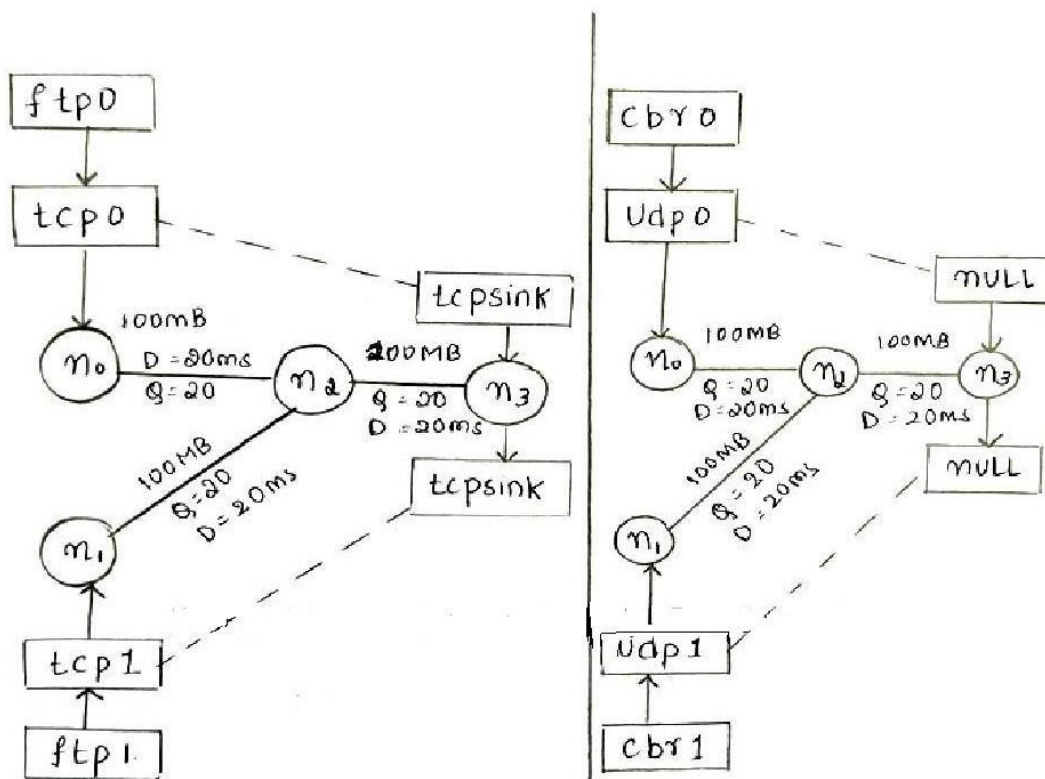


Fig.1 Point to point network with TCP

Fig.1a Point to point network with UDP

Program:

```
#=====
#   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 lab1.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open lab1.nam w]
$ns namtrace-all $namfile

#=====
#   Nodes Definition
#=====
#Create 4 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#=====
#   Links Definition
#=====
#Createlinks between nodes
$ns duplex-link $n0 $n2 300.0Mb 10ms DropTail
$ns queue-limit $n0 $n2 10
$ns duplex-link $n1 $n2 400.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 20
$ns duplex-link $n2 $n3 10.0Mb 10ms DropTail
$ns queue-limit $n2 $n3 3

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

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

```

$ns attach-agent $n3 $sink2
$ns connect $tcp0 $sink2
$tcp0 set packetSize_ 1500
#Setup a TCP connection set tcp1 [new Agent/TCP]

```

```

$ns attach-agent $n1 $tcp1
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $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"
$ns at 2.0 "$ftp0 stop"

```

```

#Setup a FTP Application over TCP connection
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 1.0 "$ftp1 start"
$ns at 2.0 "$ftp1 stop"

```

```

#=====
#   Termination
#=====
#Define a 'finish' procedure
proc finish { } {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam lab1.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

```

Nam result:

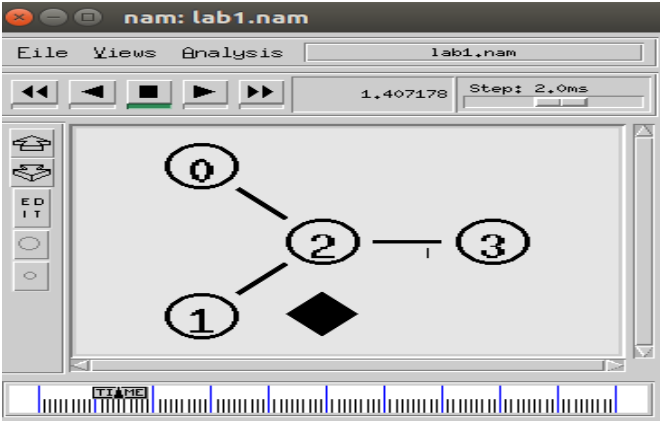


Fig. 2 Animated Results

Bandwidth(MB) N0-N2, N1-N2, N2-N3	Queue Size Q1, Q2, Q3	Received at Node 3	Dropped at Node 2
100,100,100	20, 20,20	6820	0
100,100,1	20, 20,2	508	45
300,300,10	50, 50,3	3021	48

Fig. 3 AWK Results

```
asha@asha: ~/ns2_programs
asha@asha:~$ pwd
/home/asha
asha@asha:~$ ls
Desktop      Downloads    Music        ns2_programs  Public       Videos
Documents    examples.desktop  nctuns      Pictures      Templates
asha@asha:~$ cd ns2_programs/
asha@asha:~/ns2_programs$ ns lab1.tcl
asha@asha:~/ns2_programs$ ns lab1.tcl
asha@asha:~/ns2_programs$ awk -f packetrx1.awk lab1.tr

total number of data packets received at Node 3: 299
total number of packets dropped at Node 2: 9
asha@asha:~/ns2_programs$
```

Fig. 4 Performance of network for various values of Bandwidth & Queue sizes

AWK File:

```
BEGIN{
tcppack=0
tcppack1=0
}
{
if ($1=="r"&&$4=="3"&&$5=="tcp"&&$6=="1540")
{
tcppack++;
}
if ($1=="d"&&$3=="2"&&$4=="3"&&$5=="tcp"&&$6=="1540")
{
tcppack1++;
}
}
END{
printf("\n total number of data packets received at Node 3: %d\n",
tcppack++);
printf("\n total number of packets dropped at Node 2: %d\n",
tcppack1++);
}
```

EXPERIMENT NO 2

Aim: . Implement a four node point to point network with links n0-n2, n1-n2 and n2-n3.

Apply TCP agent between n0-n3 and UDP between n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP

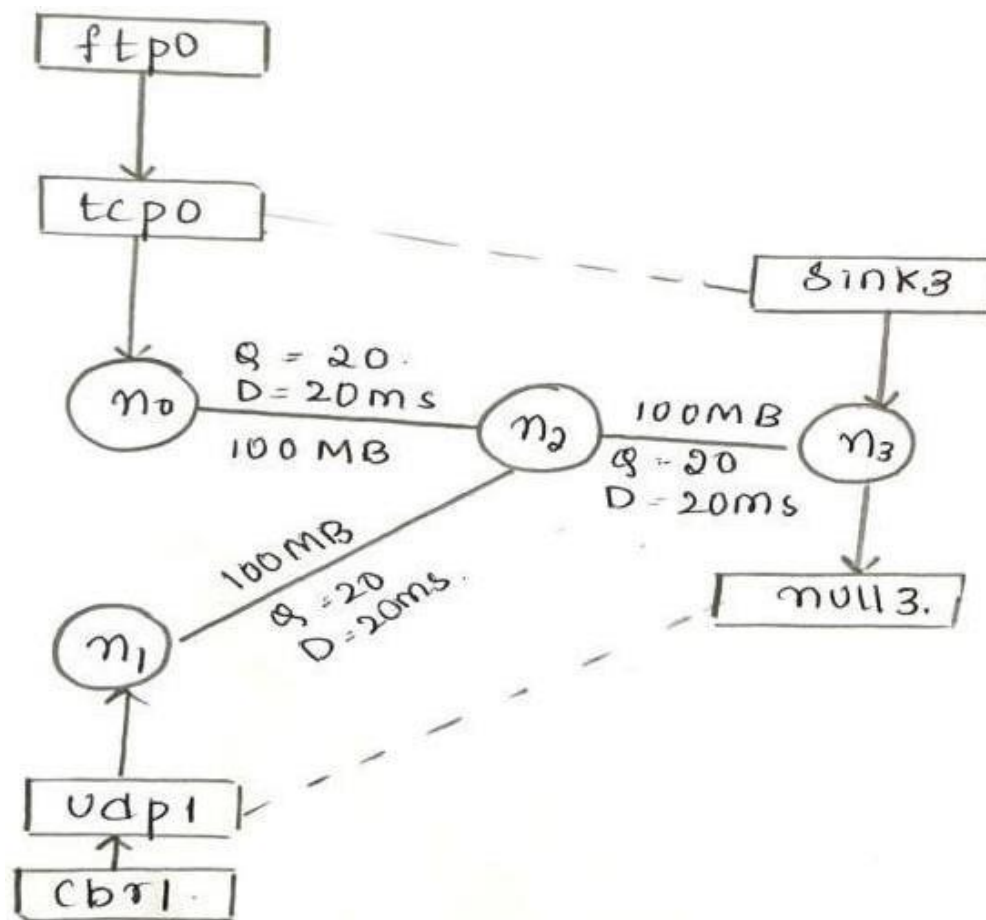


Fig. 1 Point to point network with UDP & TCP agent

```
#=====
#   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 lab2.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open lab2.nam w]
$ns namtrace-all $namfile

#=====
#   Nodes Definition
#=====
#Create 4 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#=====
#   Links Definition
#=====
#Createlinks between nodes
$ns duplex-link $n0 $n2 200.0Mb 10ms DropTail
$ns queue-limit $n0 $n2 50
$ns duplex-link $n2 $n3 200.0Mb 10ms DropTail
$ns queue-limit $n2 $n3 50
$ns duplex-link $n1 $n2 200.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 50

#Give node position (for NAM)

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
```

\$ns duplex-link-op \$n1 \$n2 orient right-up

#=====

Agents Definition

#=====

#Setup a TCP connection

set tcp0 [new Agent/TCP]

\$ns attach-agent \$n0 \$tcp0

set sink3 [new Agent/TCPSink]

\$ns attach-agent \$n3 \$sink3

\$ns connect \$tcp0 \$sink3

\$tcp0 set packetSize_ 1000

\$tcp0 set interval_ 0.1

#Setup a UDP connection

set udp1 [new Agent/UDP]

\$ns attach-agent \$n1 \$udp1

set null2 [new Agent/Null]

\$ns attach-agent \$n3 \$null2

\$ns connect \$udp1 \$null2

\$udp1 set packetSize_ 1100

\$udp1 set interval_ 0.1

#=====

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 9.0 "\$ftp0 stop"

#Setup a CBR Application over UDP connection

set cbr1 [new Application/Traffic/CBR]

\$cbr1 attach-agent \$udp1

\$cbr1 set packetSize_ 1000

\$cbr1 set rate_ 1.0Mb

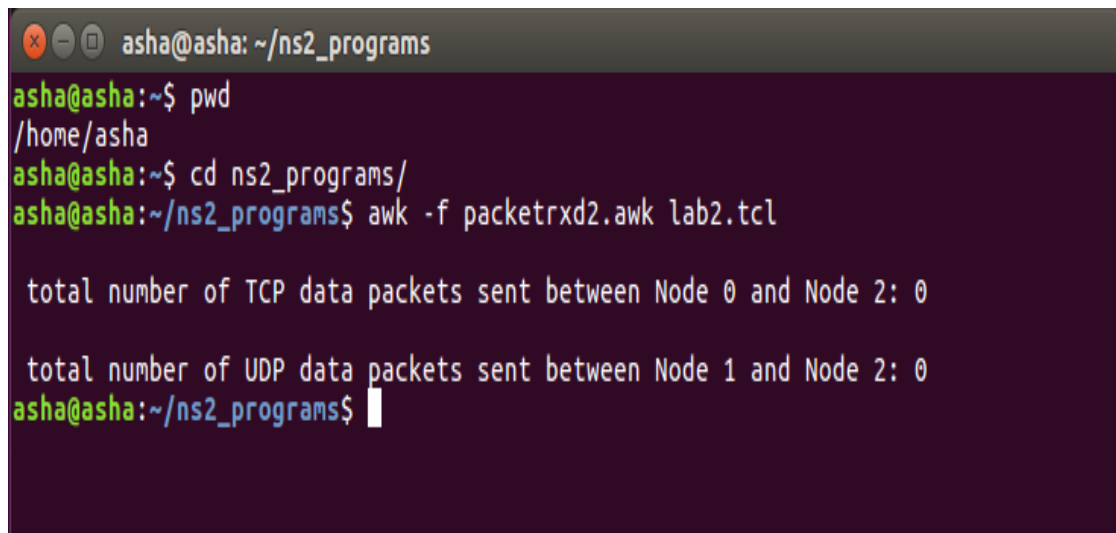
\$cbr1 set random_ null

\$ns at 1.0 "\$cbr1 start"

\$ns at 9.0 "\$cbr1 stop"

```
#=====
#   Termination
#=====
#Define a 'finish' procedure

proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam lab2.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
```

Result:

```
asha@asha: ~/ns2_programs
asha@asha:~$ pwd
/home/asha
asha@asha:~$ cd ns2_programs/
asha@asha:~/ns2_programs$ awk -f packetrx2.awk lab2.tcl

total number of TCP data packets sent between Node 0 and Node 2: 0

total number of UDP data packets sent between Node 1 and Node 2: 0
asha@asha:~/ns2_programs$
```

Fig. 2 AWK Result

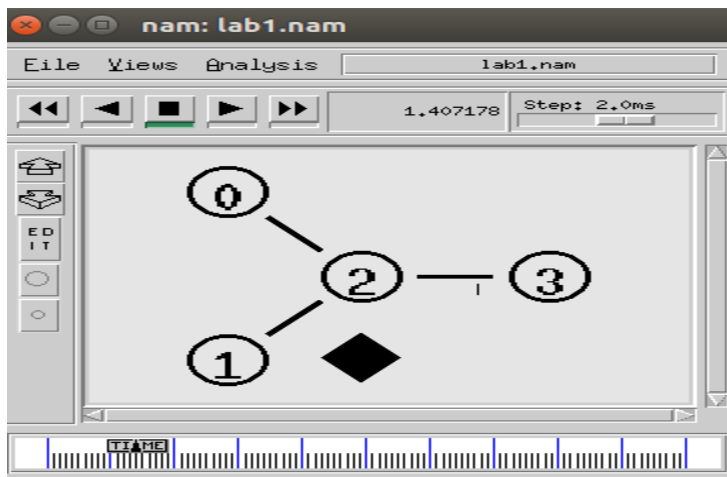


Fig. 3 Animation Result

Simulation Time (Sec)	Packet Size (Byte)	Sent at Node 2
Start Time:1	TCP:1500	TCP:3930
Stop Time:9	UDP:1500	UDP:1001
Start Time:1	TCP:1000	TCP:9410
Stop Time:20	UDP:1500	UDP:1584
Start Time:1	TCP:1300	TCP:11910
Stop Time:25	UDP:1500	UDP:2000

Fig. 4 Packet reception at node 2

```

BEGIN{
tcppack=0
tcppack1=0
}
{
if($1=="r"&&$4=="2"&&$5=="tcp"&&$6=="340")
{
tcppack++;
}
if($1=="r"&&$4=="2"&&$5=="cbr"&&$6=="300")
{
tcppack1++;
}
}
END{
printf("\n total number of TCP data packets sent between Node 0 and Node 2: %d\n", tcppack++);
printf("\n total number of UDP data packets sent between Node 1 and Node 2: %d\n", tcppack1++);
}

```

EXPERIMENT NO 3

Aim: . Implement Ethernet LAN using n (6-10) nodes. Compare the throughput by changing the error rate and data rate.

Topology:

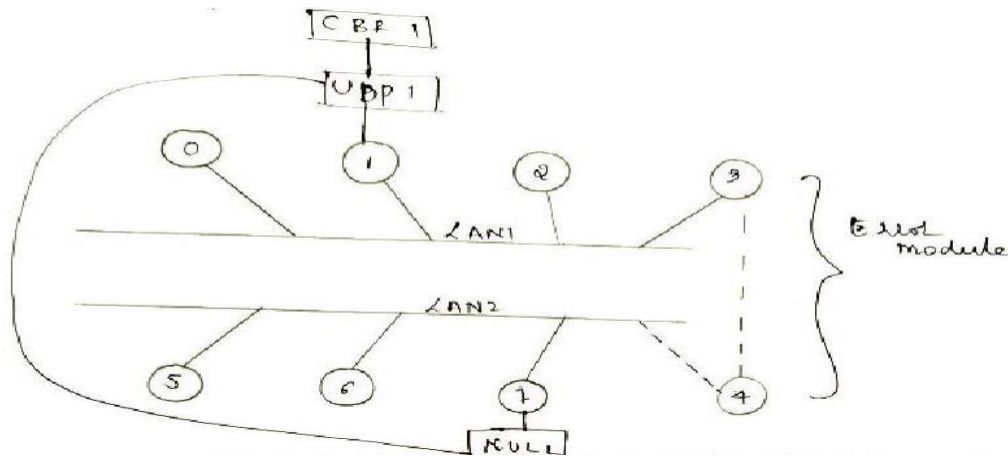


Fig.1. Ethernet LAN of 6 nodes

Program:

```
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf
```

```
set nf [open lab3.nam w]
$ns namtrace-all $nf
```

```
$ns color 0 blue
```

```
set n0 [$ns node]
$n0 color "red"
set n1 [$ns node]
$n1 color "red"
set n2 [$ns node]
$n2 color "red"
set n3 [$ns node]
```

```
$n3 color "red"
set n4 [$ns node]
$n4 color "magenta"
```

```
set n5 [$ns node]
$n5 color "magenta"
set n6 [$ns node]
$n6 color "magenta"
set n7 [$ns node]
$n7 color "magenta"
```

```
$n1 label "Source/UDP"
$n3 label "Error Node"
$n7 label "Destination"
```

```
$ns make-lan "$n0 $n1 $n2 $n3" 100Mb 300ms LL Queue/DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n6 $n7" 100Mb 300ms LL Queue/DropTail Mac/802_3
```

```
$ns duplex-link $n3 $n4 100Mb 300ms DropTail
$ns duplex-link-op $n3 $n4 color "green"
```

```
# set error rate. Here ErrorModel is a class and it is single word and space should not
be given between Error and Model
# lossmodel is a command and it is single word. Space should not be given between
loss and model
```

```
set err [new ErrorModel]
$ns lossmodel $err $n3 $n4
$err set rate_ 0.3
```

```
# error rate should be changed for each output like 0.1,0.3,0.5.... */
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
```

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
```

```
$cbr set fid_ 0
$cbr set packetSize_ 1000
```

```
$cbr set interval_ 0.1
set null [new Agent/Null]
$ns attach-agent $n7 $null
```

```
$ns connect $udp $null
```

```
proc finish { } {
```

```

global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab3.nam &
exit 0
}

```

```

$ns at 0.1 "$cbr start"
$ns at 3.0 "finish"
$ns run

```

Nam result:

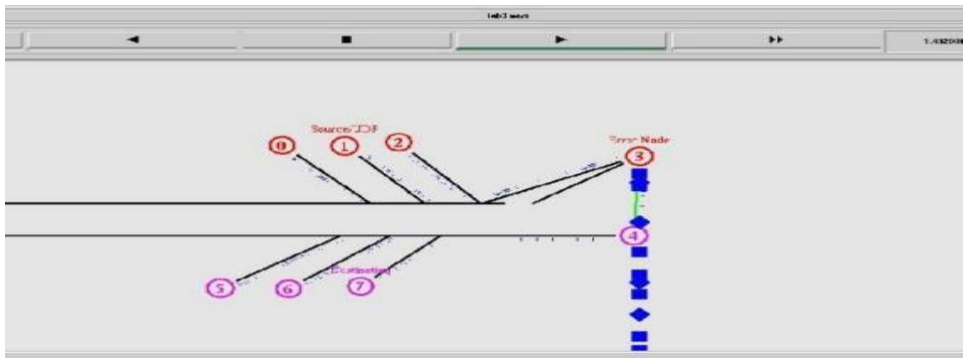


Fig. 2 Animation result

```

asha@asha: ~/ns2_programs
asha@asha:~/ns2_programs$ awk -f packetrx3.awk lab3.tr

total number of data packets at Node 7: 8
asha@asha:~/ns2_programs$

```

Fig. 3 AWK result

Error Rate	Data Rate	Received at Node 7
0.1	0.001	1255
0.3	0.01	95
0.5	0.1	05

Fig.4 Table to show the packet reception at node 7

AWK file:

```
BEGIN{
tcppack=0
tcppack1=0
}
{
if($1=="r"&&$4=="7"&&$5=="cbr"&&$6=="1000")
{
tcppack++;
}
}
END{
printf("\n total number of  data packets at Node 7: %d\n", tcppack++);

}
```

EXPERIMENT NO 4

Aim: . Implement Ethernet LAN using n nodes and assign multiple traffic to the nodes and obtain congestion window for different sources/ destinations.

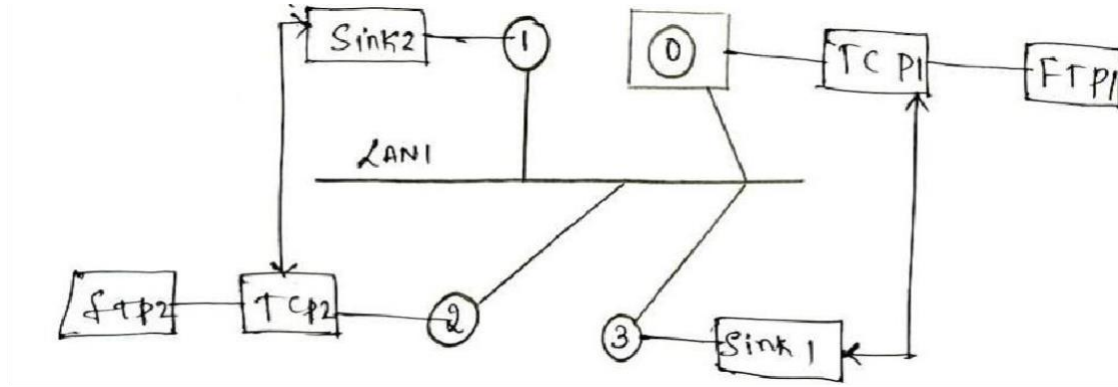


Fig.1 Ethernet LAN to show congestion window

Program:

```
set ns [new Simulator]
set tf [open lab4.tr w]
$ns trace-all $tf
```

```
set nf [open lab4.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
```

```
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

```
$ns make-lan "$n0 $n1 $n2 $n3" 10mb 10ms LL Queue/DropTail Mac/802_3
```

```
set tcp0 [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp0
```

```
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

```
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
```

```
$ns connect $tcp0 $sink3
```

```
set tcp2 [new Agent/TCP]

$ns attach-agent $n2 $tcp2

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp2 $sink1

#####To trace the congestion window#####
set file1 [open file1.tr w]
$tcp0 attach $file1
$tcp0 trace cwnd_

#$tcp0 set maxcwnd_ 10
set file2 [open file2.tr w]
$tcp2 attach $file2
$tcp2 trace cwnd_
proc finish { } {
    global nf tf ns
    $ns flush-trace
    exec nam lab7.nam &
    close $nf
    close $tf
    exit 0
}
$ns at 0.1 "$ftp0 start"
$ns at 1.5 "$ftp0 stop"
$ns at 2 "$ftp0 start"
$ns at 3 "$ftp0 stop"
$ns at 0.2 "$ftp2 start"
$ns at 2 "$ftp2 stop"
$ns at 2.5 "$ftp2 start"
$ns at 4 "$ftp2 stop"
$ns at 5.0 "finish"
$ns run
```

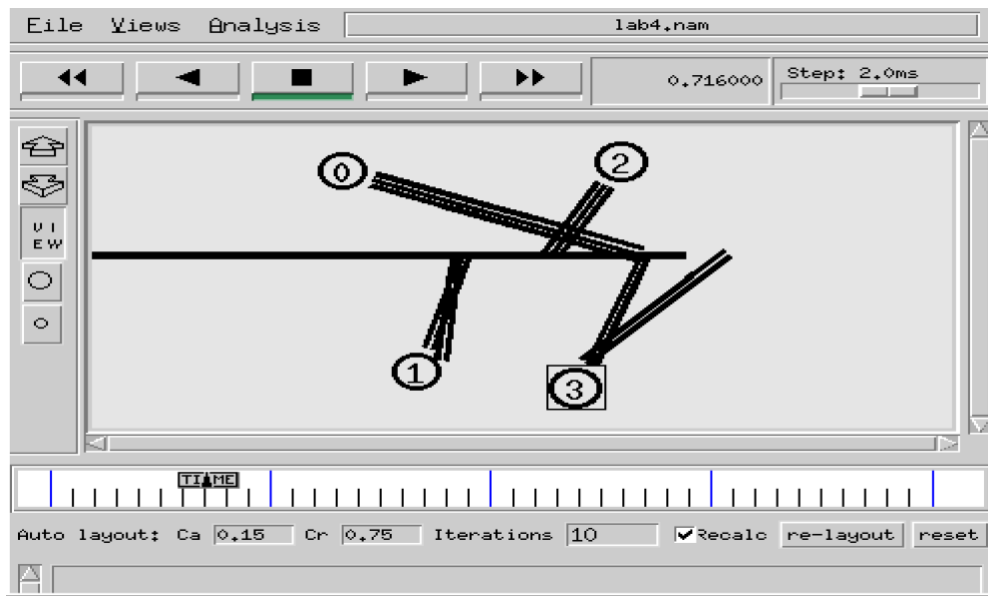
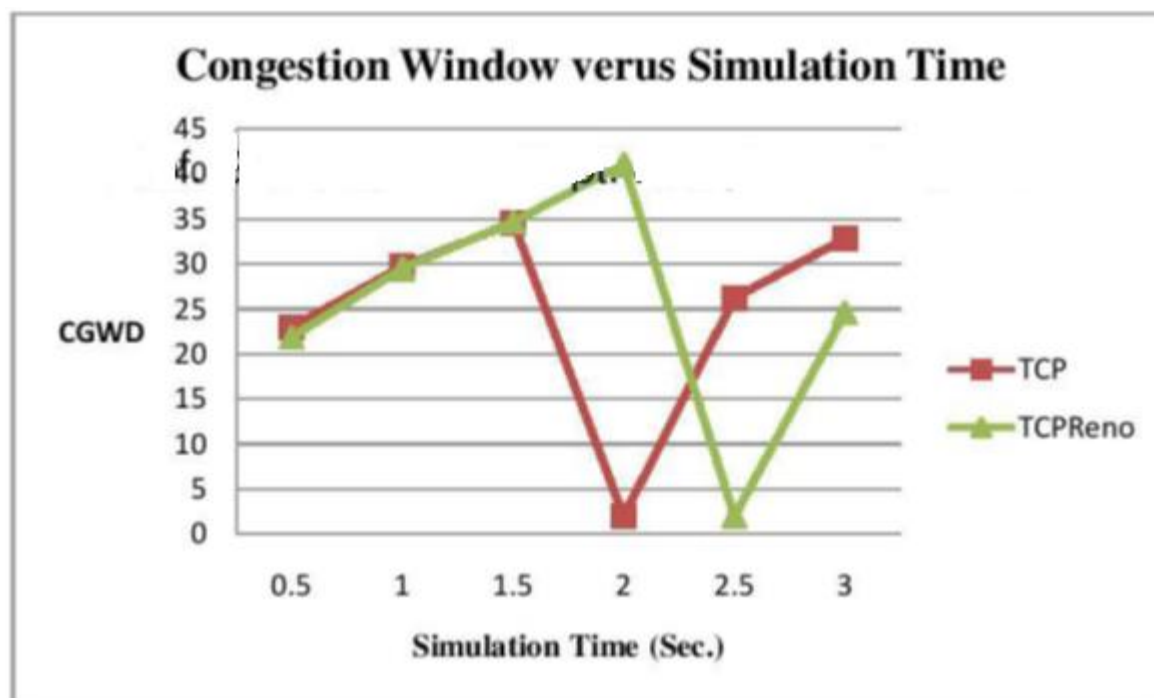
Nam result:

Fig. 2 Animation result

Time	TCP	TCPReno
0.5	23.02	21.91
1	29.73	29.53
1.5	34.59	34.71
2	2	41.11
2.5	26.23	2
3	32.84	24.62

Fig. 3 Congestion for TCP & TCP Reno



EXPERIMENT NO 5

Aim: Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters.

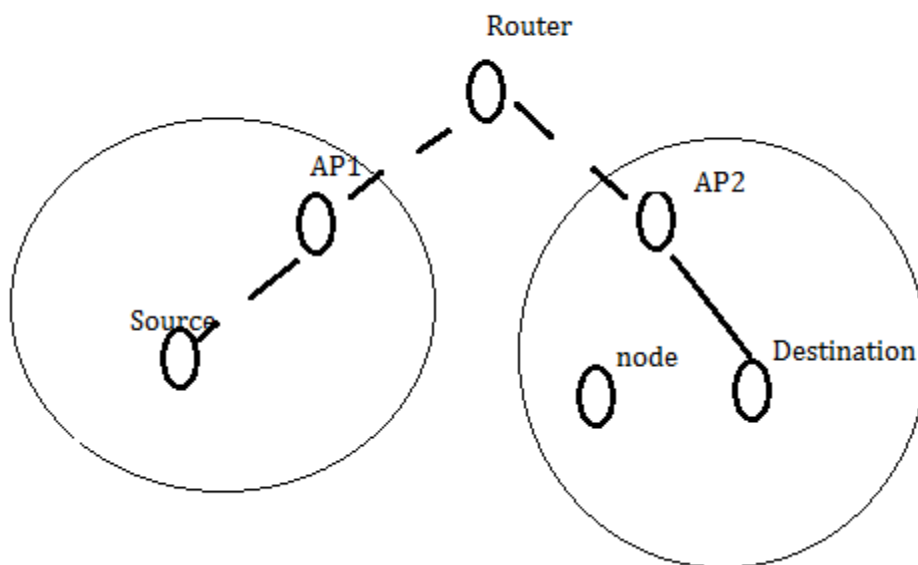


Fig. 1 Wireless LAN

Program:

```
set ns [new Simulator]
set tf [open expt55.tr w]
$ns trace-all $tf

set topo [new Topography]
$topo load_flatgrid 1000 1000
set nf [open expt55.nam w]
$ns namtrace-all-wireless $nf 2000 2000

set chan [new Channel/WirelessChannel];#Create wireless channel

$ns node-config -adhocRouting AODV \
    -llType LL \
    -macType Mac/802_11 \
    -ifqType Queue/DropTail \
    -ifqLen 50 \
    -phyType Phy/WirelessPhy \
    -channel $chan \
```

```
-propType Propagation/TwoRayGround \  
-antType Antenna/OmniAntenna \  
-topoInstance $topo \  
-agentTrace ON\  
-routerTrace ON \  
-macTrace ON
```

```
create-god 6  
set n0 [$ns node]  
set n1 [$ns node]  
set n2 [$ns node]  
set n3 [$ns node]  
set n4 [$ns node]  
#set n5 [$ns node]  
set n6 [$ns node]  
#set n7 [$ns node]
```

```
$n0 label "tcp-Source"  
$n1 label "Access Point1"  
$n2 label "Router"  
$n3 label "Access Point2"  
$n4 label "Destination"  
#$n5 label "node1"  
$n6 label "node2"  
#$n7 label "gateway"
```

```
$n0 set X_ 10  
$n0 set Y_ 50  
$n0 set Z_ 0  
$ns initial_node_pos $n0 20
```

```
$n1 set X_ 120  
$n1 set Y_ 130  
$n1 set Z_ 0  
$ns initial_node_pos $n1 20  
$n2 set X_ 200  
$n2 set Y_ 230  
$n2 set Z_ 0  
$ns initial_node_pos $n2 20  
$n3 set X_ 300  
$n3 set Y_ 130  
$n3 set Z_ 0  
$ns initial_node_pos $n3 20  
$n4 set X_ 350  
$n4 set Y_ 20  
$n4 set Z_ 0  
$ns initial_node_pos $n4 20
```

```
$n6 set X_ 600
$n6 set Y_ 20
$n6 set Z_ 0
$ns initial_node_pos $n6 20

$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n4 setdest 900 50 20"
```

```
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

```
set sink4 [new Agent/TCPSink]
$ns attach-agent $n4 $sink4
$ns connect $tcp0 $sink4
```

```
$ns at 5 "$ftp0 start"
$ns at 50 "$ftp0 stop"
```

```
#=====
#   Termination
#=====
proc finish { } {
    global ns nf tf
    $ns flush-trace
    exec nam expt55.nam &
    close $tf
    exit 0
}
$ns at 80 "finish"
$ns run
```

Nam result:

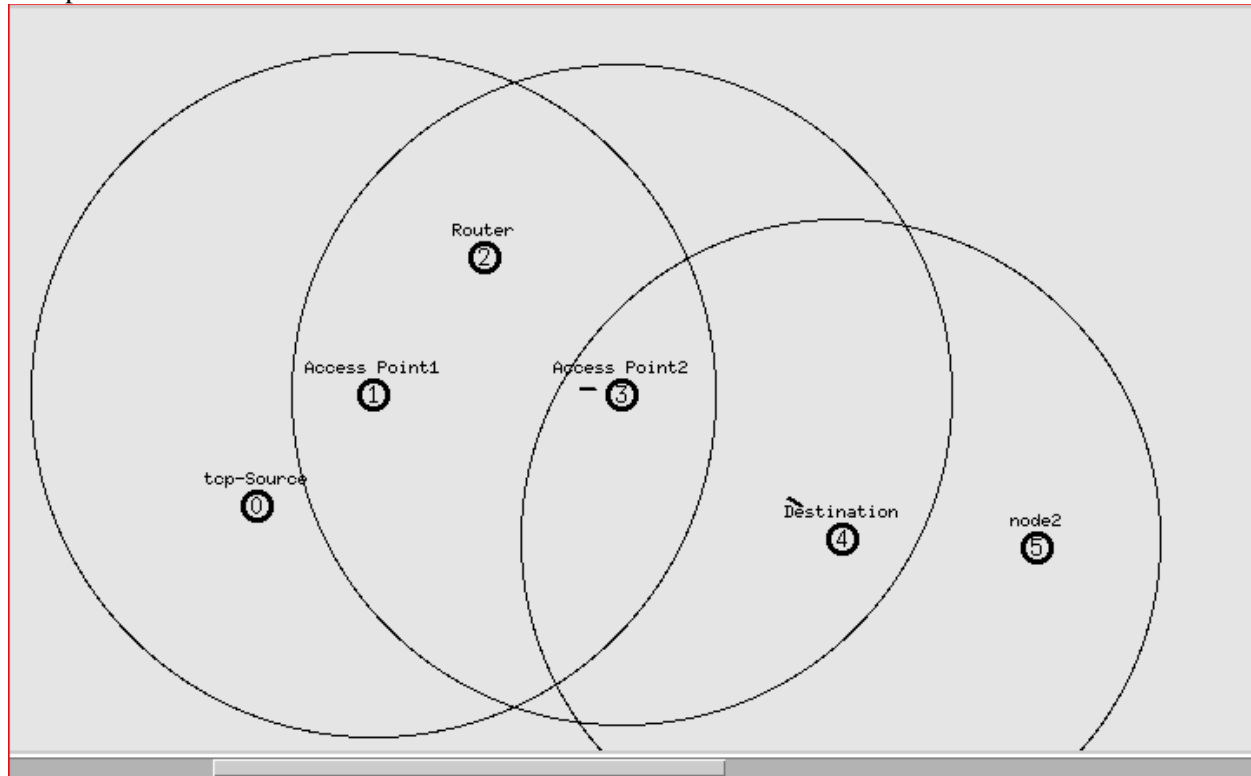


Fig. 2 Animation result

```

BEGIN{
cbrpack=0
cbrpack1=0
}
{
if($1=="r"&&$4=="AGT")
{
cbrpack++;
}
if($1=="s"&&$4=="AGT")
{
cbrpack1++;
}
}

END{
printf("\n total number of packets sent: %d\n", cbrpack1++);
printf("\n total number of packets received: %d\n", cbrpack++);
}

```

EXPERIMENT NO 6

Aim: Implementation of Link state routing algorithm.

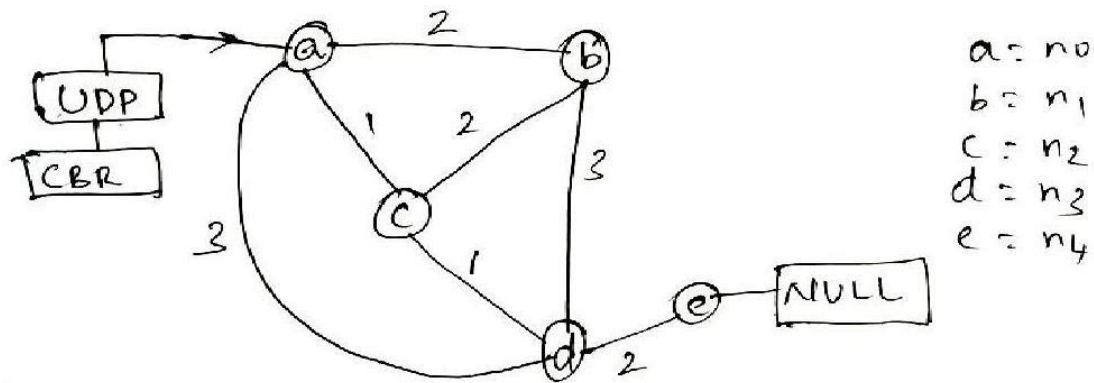


Fig. 1 Topology with 5 nodes

Program:

```

#=====
#   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 lab6.tr w]
$ns trace-all $tracefile

#Open the NAM trace file

```

```

set namfile [open lab6.nam w]
$ns namtrace-all $namfile
#=====
#   Nodes Definition
#=====
#Create 5 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

#=====
#   Links Definition
#=====
#Createlinks between nodes
$ns duplex-link $n0 $n1 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n1 50
$ns duplex-link $n0 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n2 50
$ns duplex-link $n2 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n2 $n3 50
$ns duplex-link $n1 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n1 $n3 50
$ns duplex-link $n3 $n4 100.0Mb 10ms DropTail
$ns queue-limit $n3 $n4 50
$ns duplex-link $n0 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n3 50
$ns duplex-link $n1 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 50

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

#Set the link costs. All link costs are symmetric

$ns cost $n0 $n1 2

$ns cost $n0 $n2 1
$ns cost $n0 $n3 3

```

```
$ns cost $n1 $n0 2
$ns cost $n1 $n2 2
$ns cost $n1 $n3 3
```

```
$ns cost $n2 $n1 2
$ns cost $n2 $n0 1
$ns cost $n2 $n3 1
```

```
$ns cost $n3 $n2 1
$ns cost $n3 $n1 3
$ns cost $n3 $n0 3
$ns cost $n3 $n4 2
```

```
$ns cost $n4 $n3 2
```

```
#=====
#   Agents Definition
#=====
#Setup a UDP connection
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set null1 [new Agent/Null]
$ns attach-agent $n4 $null1
$ns connect $udp0 $null1
$udp0 set packetSize_ 1500
```

```
#=====
#   Applications Definition
#=====
#Setup a CBR Application over UDP connection
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packetSize_ 1000
$cbr0 set rate_ 1.0Mb
$cbr0 set random_ null
$ns at 1.0 "$cbr0 start"
$ns at 5.0 "$cbr0 stop"
```

```
$ns rtproto LS
```

```
#=====
#   Termination
#=====
```



```
#Define a 'finish' procedure
proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam lab6.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
```

Nam result:

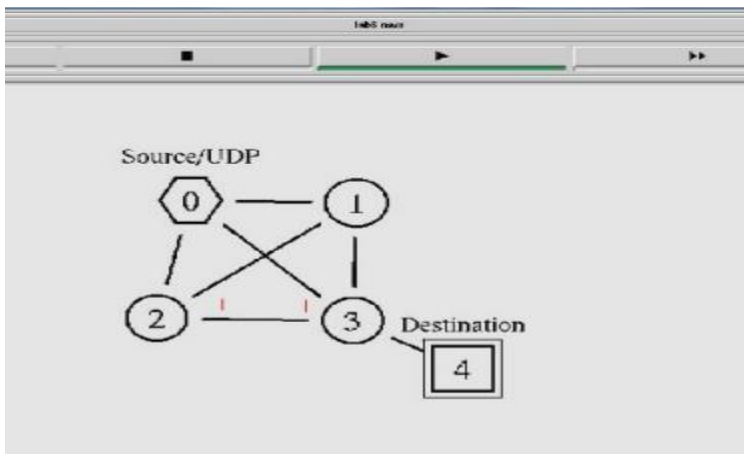


Fig. 2 Animation result

AWk file:

```
BEGIN{
    tcppack=0
    tcppack1=0
}
{
    if($1=="r"&&$4=="4"&&$5=="cbr"&&$6=="1000")
    {
        tcppack++;
    }
}
END{
    printf("\n total number of  data packets at Node 4 due to Link state algorithm: %d\n", tcppack++);
}
```

PART-B

EXPERIMENTS

USING DEV- C ++

EXPERIMENT NO 1 A

Aim: Write a program for a HDLC frame to perform the following.

i) Bit stuffing


Theory: The functions of data link layer are Data link control and multiple access control. The main function of Data link control is framing. Data Link Layer adds source and destination MAC address and pack the data into frame.

There are two types of framing, fixed size & Variable size framing. In Fixed size, the size itself acts as a delimiter. In Variable size we need a way to define the end of the frame and the beginning of the next. Hence to identify the end and the beginning of the frame DLL adds flag (01111110). In the data a '0' bit is automatically stuffed into the outgoing bit stream whenever the sender's data link layer finds five consecutive 1s. This bit stuffing is similar to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.

When the receiver sees five consecutive incoming '1' bits, followed by a '0' bit, it automatically de-stuffs (i.e., deletes) the 0 bit. Bit Stuffing is completely transparent to network layer as byte stuffing. The figure 1 below gives an example of bit stuffing

For example if the data bits are 011011011111101101100 and flag is 01111110 then after bit stuffing the data will be

01111110 1101101111101100 01111110



```
#include<stdio.h>
#include<string.h>
#include<conio.h>
```

```
main()
{
    char a[20],fs[50]="",t[6],r[5];
    int i,j,p=0,q=0;
    printf("enter bit string : ");
    scanf("%s",a);
    strcat(fs,"01111110");
    if(strlen(a)<5)
    {
        strcat(fs,a);
    }

    else
    {
        for(i=0;i<strlen(a)-4;i++)
```

```
{
    for(j=i;j<i+5;j++)
    {
        t[p++]=a[j];
    }
    t[p]='\0';
    if(strcmp(t,"11111")==0)
    {
        strcat(fs,"111110");
        i=j-1;
    }
    else
    {
        r[0]=a[i];
        r[1]='\0';
        strcat(fs,r);
    }
    p=0;
}
for(q=i;q<strlen(a);q++)
{
    t[p++]=a[q];
}
t[p]='\0';
strcat(fs,t);
}
strcat(fs,"01111110");
printf("After stuffing : %s",fs);
getch();
}
```

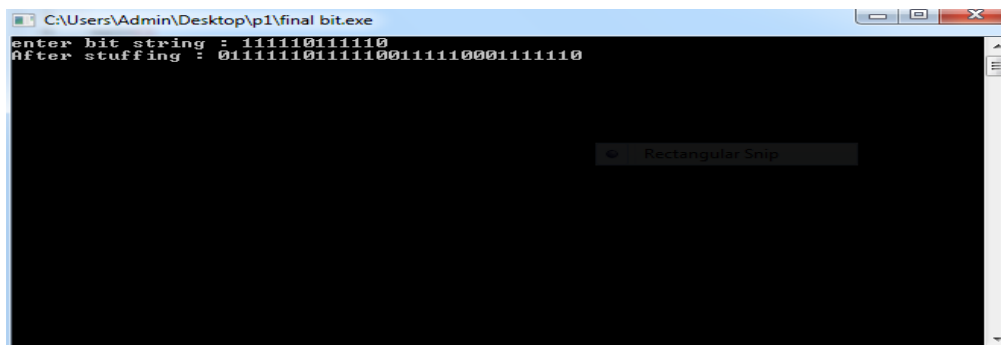
Result:

Fig1. Results of bit stuffing

EXPERIMENT NO 1 B

Aim: Write a program for a HLDC frame to perform the following.

ii) Character stuffing.

Theory:

In character stuffing an escape byte(ESC) is added in the data as shown below.If

bmsitece is the data and a is the beginning flag is the data and b is the end flag then the data after character stuffing will be

abb**msitece**b****


Program:

```
#include<string.h>
#include<stdio.h>
#include<conio.h>
main()
{
    char a[30],fs[50000]="",t[3],sd,ed,x[3],s[3],d[3],y[3];
    int i,j,p=0,q=0;

    printf("Enter characters to be stuffed : ");
    scanf("%s",a);
    printf("\nEnter a character that represents starting delimiter : ");
    scanf(" %c",&sd);
    printf("\nEnter a character that represents ending delimiter : ");
    scanf(" %c",&ed);
    x[0]=s[0]=s[1]=sd;
    x[1]=s[2]='\0';
    y[0]=d[0]=d[1]=ed;
    d[2]=y[1]='\0';
    strcat(fs,x);

    for(i=0;i<strlen(a);i++)
    {
        t[0]=a[i];
        t[1]='\0';
        if(t[0]==sd)

            strcat(fs,s);
```

```
else
    if (t[0]==ed)
        strcat(fs,d);
else
    strcat(fs,t);

}

    strcat(fs,y);
printf("\n after stuffing : %s",fs);
getch();
}
```

Result:

```
\\128.0.33.14\\NS2Softwares\\NS 2.35 data\\workshop\\fdpccn\\p3\\final char.exe
Enter characters to be stuffed : dbit
Enter a character that represents starting delimiter : a
Enter a character that represents ending delimiter : b
After stuffing : adbbith
```

Fig.2 Results of byte stuffing.

EXPERIMENT NO 2

Aim: Write a program for distance vector algorithm to find suitable path for transmission.

Theory: Routing algorithms can be grouped into static and dynamic. Static routing algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Instead, the choice of route from one node to another is to use to computed in advance, offline, and downloaded to the routers when the network ids booted.

Dynamic or Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every T sec, when the load changes, or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time)

Two algorithms in particular, distance vector routing and link state routing are the most popular. Distance vector routing algorithms operate by having each router maintain a table (i.e., vector) giving the best known distance to each destination and which line to get there. These tables are updated by exchanging information with the neighbors.

The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm. It was the original ARPANET routing algorithm and was also used in the Internet under the RIP and in early versions of DECnet and Novell's IPX. AppleTalk and Cisco routers use improved distance vector protocols.

Distance vector mainly has three steps.

Initialization: The nodes which are directly connected to the source will have a finite entry in the table and the nodes which are not directly connected will have infinity entry.

Sharing: If there is a presence of an intermediate node between source and destination the intermediate node shares its routing table to help the source find the efficient path.

Updating: Once the source uses the routing table of neighbor, it updates its routing table.

Program :

```
#include <stdio.h>
```

```
struct node
```

```
{
```

```
    unsigned dist[20];
```

```
    unsigned from[20];
```

```
    }rt[10];

int main()
{
    int costmat[20][20],source,desti;
    int nodes,i,j,k,count=0;
    printf("\nEnter the number of nodes : ");
    scanf("%d",&nodes);//Enter the nodes
    printf("\nEnter the cost matrix :\n");
    for(i=0;i<nodes;i++)
    for(j=0;j<nodes;j++)
    {
        scanf("%d",&costmat[i][j]);
        costmat[i][i]=0;
        rt[i].dist[j]=costmat[i][j];
        rt[i].from[j]=j;
    }
    for(i=0;i<nodes;i++)
    {
        printf("\n\n For router %d\n",i);
        for(j=0;j<nodes;j++)
        printf("\t\tnode %d via %d Distance %d",j,rt[i].from[j],rt[i].dist[j]);
    }
    do
    {
        count=0;
        for(i=0;i<nodes;i++)
        for(j=0;j<nodes;j++)
        // if(i!=j)
        for(k=0;k<nodes;k++)
        if(rt[i].dist[j]>rt[i].dist[k]+rt[k].dist[j])
        {
            rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
            rt[i].from[j]=rt[i].from[k];
            count++;
        }
    }while(count!=0);
    for(i=0;i<nodes;i++)
    {
        printf("\n\n For router %d\n",i+1);
        for(j=0;j<nodes;j++)
        printf("\t\tnode%d via %d Distance %d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
    }
    printf("\n\n");
}
```


Result:

Result of initialization:

```
C:\Users\Asha\Desktop\Untitled1.exe

Enter the number of nodes : 5
Enter the cost matrix :
0 3 2 999 999
3 0 999 1 999
2 999 0 2 999
999 1 2 0 5
999 999 999 5 0

For router 0
node 0 via 0 Distance 0
node 1 via 1 Distance 3
node 2 via 2 Distance 2
node 3 via 3 Distance 999
node 4 via 4 Distance 999

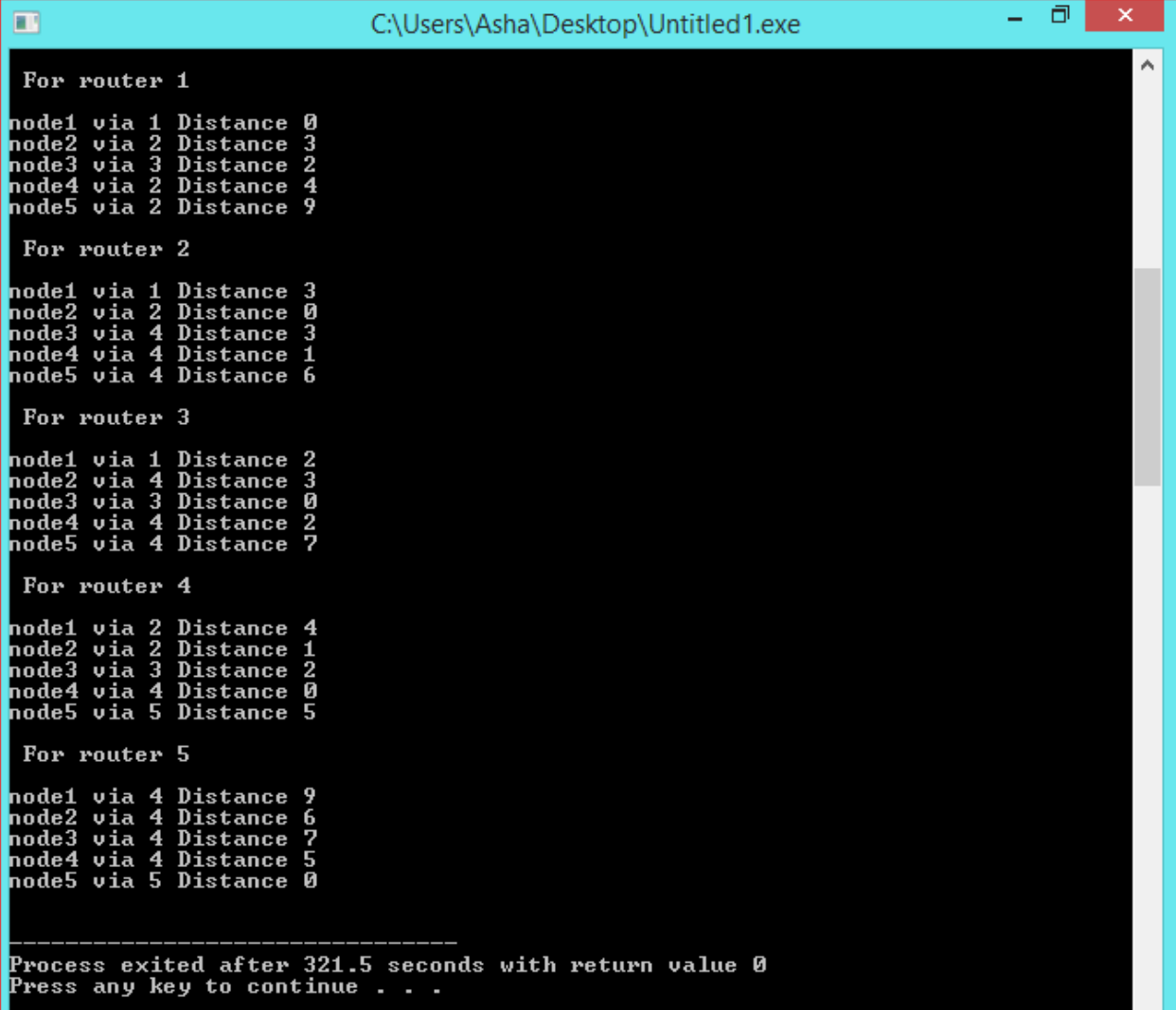
For router 1
node 0 via 0 Distance 3
node 1 via 1 Distance 0
node 2 via 2 Distance 999
node 3 via 3 Distance 1
node 4 via 4 Distance 999

For router 2
node 0 via 0 Distance 2
node 1 via 1 Distance 999
node 2 via 2 Distance 0
node 3 via 3 Distance 2
node 4 via 4 Distance 999

For router 3
node 0 via 0 Distance 999
node 1 via 1 Distance 1
node 2 via 2 Distance 2
node 3 via 3 Distance 0
node 4 via 4 Distance 5

For router 4
node 0 via 0 Distance 999
node 1 via 1 Distance 999
node 2 via 2 Distance 999
node 3 via 3 Distance 5
node 4 via 4 Distance 0
```

Final routing tables of each node in the network:



```
For router 1
node1 via 1 Distance 0
node2 via 2 Distance 3
node3 via 3 Distance 2
node4 via 2 Distance 4
node5 via 2 Distance 9

For router 2
node1 via 1 Distance 3
node2 via 2 Distance 0
node3 via 4 Distance 3
node4 via 4 Distance 1
node5 via 4 Distance 6

For router 3
node1 via 1 Distance 2
node2 via 4 Distance 3
node3 via 3 Distance 0
node4 via 4 Distance 2
node5 via 4 Distance 7

For router 4
node1 via 2 Distance 4
node2 via 2 Distance 1
node3 via 3 Distance 2
node4 via 4 Distance 0
node5 via 5 Distance 5

For router 5
node1 via 4 Distance 9
node2 via 4 Distance 6
node3 via 4 Distance 7
node4 via 4 Distance 5
node5 via 5 Distance 0

-----
Process exited after 321.5 seconds with return value 0
Press any key to continue . . .
```

Fig. 1 Results of “Distance Vector algorithm”

EXPERIMENT NO 3

Aim: Implement Dijkstra's algorithm to compute the shortest routing path.

Theory:

Routing algorithms are again classified as Intra domain and Inter domain. Dijkstra algorithm is the inter domain routing algorithm. This algorithm is used to find the shortest path between source and destination.

The shortest path is a graph of the network with each node representing router & communication link by an arc of the graph. The purpose of the algorithm is to find the shortest path. The shortest path may be in terms of hops or of geographical distance in Kms or of the mean queuing & transmission delay. Generally the labels on the arc are computed as a function of distance, Bandwidth, cost, average traffic, mean queue length, delay and other factors.

Dijkstra's method of computing the shortest path is a static routing algorithm. It involves building a graph of the subnet, with each node of the graph representing a router and each arc representing a communication line or a link. To find a route between a pair of routers, the algorithm just finds the shortest path between them on the graph.

Dijkstra's algorithm finds the solution for the shortest path problems only when all the edge-weights are non-negative on a weighted, directed graph. In Dijkstra's algorithm the metric used for calculation is distance. Each node is labeled with its distance from the source node along the best known path. Initially, no paths are known, so all nodes are labeled with infinity. As the algorithm proceeds and path are found, the labels may change, reflecting better paths. A label may either be tentative or permanent. Initially all nodes are tentative and once it is discovered that the shortest possible path to a node is got it is made permanent and never be changed.

Program:

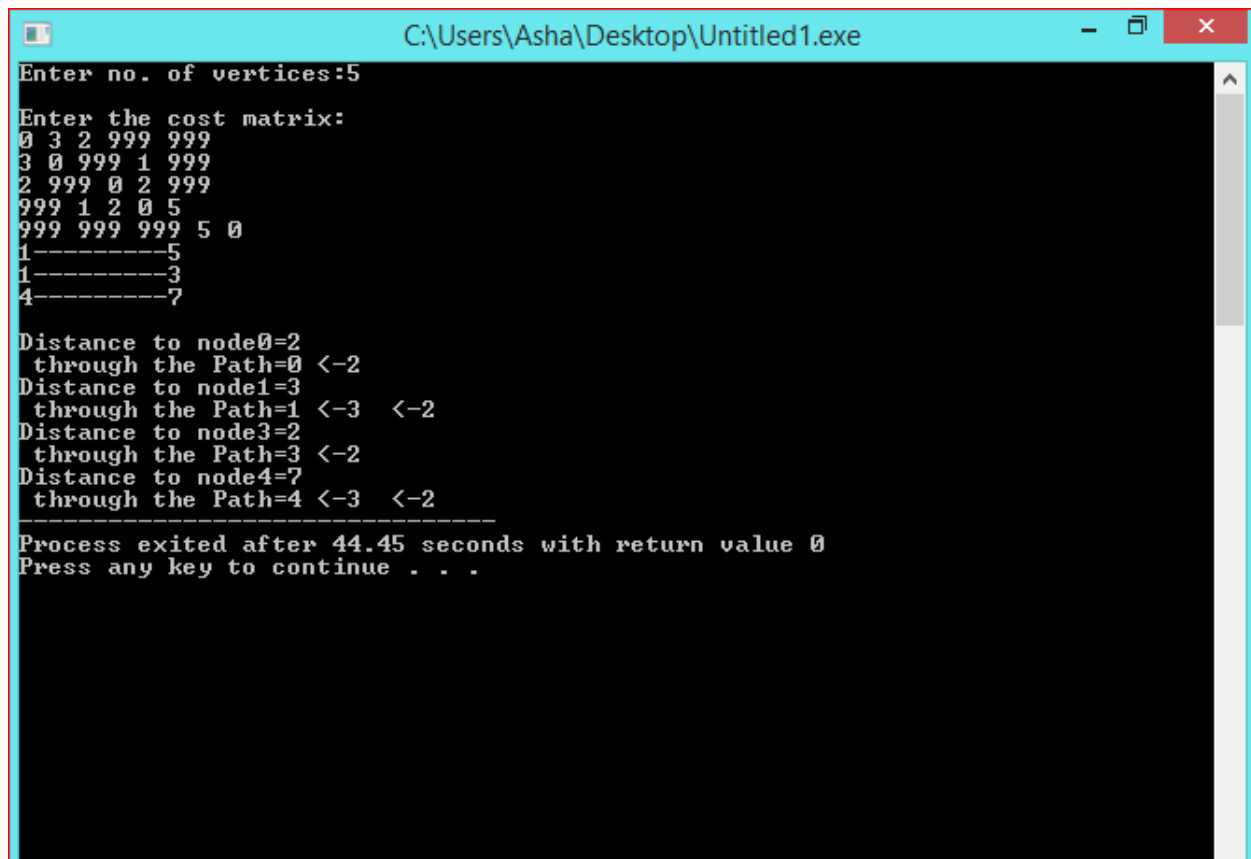
```
#include<stdio.h>
#include<conio.h>
#define INFINITY 99
#define startnode 2

void dijkstra(int cost[10][10],int n);
int main()
{   int cost[10][10],i,j,n,u;
    printf("Enter no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the cost matrix:\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&cost[i][j]);
    dijkstra(cost,n);
    return 0;
}
```

```

void dijkstra(int cost[10][10],int n)
{
    int distance[10],pred[10];
    int visited[10],count, mindistance, nextnode, i, j;    //initialize pred[],distance[] and visited[]
    for(i=0;i<n;i++)
    {
        distance[i]=cost[startnode][i];
        pred[i]=startnode;
        visited[i]=0;
    }
    distance[startnode]=0;
    visited[startnode]=1;
    count=1;
    while(count<n-1)
    {
        mindistance=INFINITY;    //nextnode gives the node at minimum distance
        for(i=0;i<n;i++)
            if(distance[i]<mindistance&&!visited[i])
            {
                mindistance=distance[i];
                nextnode=i;
            }
        //check if a better path exists through nextnode
        visited[nextnode]=1;
        for(i=0;i<n;i++)
            if(!visited[i])
                if(mindistance+cost[nextnode][i]<distance[i])
                {
                    distance[i]=mindistance+cost[nextnode][i];
                    printf("%d-----%d\n",i,distance[i]) ;
                    pred[i]=nextnode;
                }
        count++;
    }
    //print the path and distance of each node
    for(i=0;i<n;i++)
        if(i!=startnode)
        {
            printf("\nDistance to node%d=%d",i,distance[i]);
            printf("\n through the Path=%d",i);
            j=i;
            do
            {
                j=pred[j];
                printf(" <-%d ",j);
            }while(j!=startnode);
        }
}

```

Result:

```
C:\Users\Asha\Desktop\Untitled1.exe
Enter no. of vertices:5
Enter the cost matrix:
0 3 2 999 999
3 0 999 1 999
2 999 0 2 999
999 1 2 0 5
999 999 999 5 0
1-----5
1-----3
4-----7

Distance to node0=2
through the Path=0 <-2
Distance to node1=3
through the Path=1 <-3 <-2
Distance to node3=2
through the Path=3 <-2
Distance to node4=7
through the Path=4 <-3 <-2
-----
Process exited after 44.45 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT NO 4

Aim: For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases

a. Without error

Theory: A cyclic redundancy check (CRC) is an error detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Data entering to this system get a short check value attached, based on remainder of polynomial division of their contents. on retrieval, the calculation is repeated, and corrective actions can be taken against the presumed data if the check value does not match.

Algorithm:-

- Given a bit string, append 0s to the end of it (the number of 0 s is the same as the degree of the generator polynomial) let $B(x)$ be the polynomial corresponding to B
- Divide $B(x)$ by some agreed on polynomial $G(x)$ (generator polynomial) and determine the remainder $R(x)$. This division is to be done using Modulo 2 Division.
- Define $T(x) = B(x) - R(x)$
- $(T(x)/G(x) \Rightarrow \text{remainder } 0)$
- Transmit T, the bit string corresponding to $T(x)$.
- Let T' represent the bit stream the receiver gets and $T'(x)$ the associated polynomial. The receiver divides $T'(x)$ by $G(x)$. If there is a 0 remainder, the receiver concludes $T = T'$ and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.

```
#include<stdio.h>
#include<string.h>

char t[30],cs[30],g[10];

int a,i,j,N;

void xor1()
{
for(j = 1;j < N; j++)
cs[j] = (( cs[j] == g[j])?'0':'1');
}

void crc()
{
for(i=0;i<N;i++)
```

```

cs[i]=t[i];
do
{
if(cs[0]=='1')
xor1();
for(j=0;j<N-1;j++)
cs[j]=cs[j+1];
cs[j]=t[i++];
}while(i<=a+N-1);
}

```

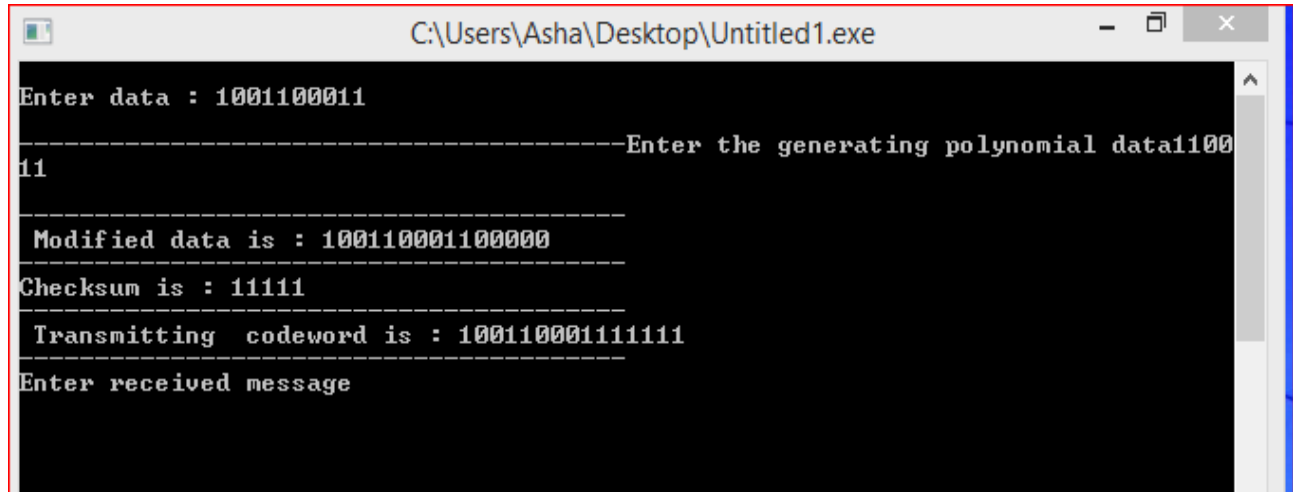
```

int main()
{
printf("\nEnter data : ");
scanf("%s",t);
printf("\n-----");
printf("Enter the generating polynomial data");
scanf("%s",g);
N=strlen(g);
a=strlen(t);

if(((N-1)< a) && (g[0]=='1') && (g[N-1]=='1') )
{
for(i=a;i<a+N-1;i++)
t[i]='0';
printf("\n-----");
printf("\n Modified data is : %s",t);
printf("\n-----");
crc();
printf("\nChecksum is : %s",cs);
for(i=a;i<a+N-1;i++)
t[i]=cs[i-a];
printf("\n-----");
printf("\n Transmitting  codeword is : %s",t);
printf("\n-----");
printf("\nEnter received message ");
scanf("%s",t);
crc();
for(i=0;(i<N-1) && (cs[i]!='1');i++);
if(i<N-1)
printf("\nError detected\n\n");
else
printf("\nNo error detected\n\n");
printf("\n-----\n");
}
else
printf("wrong generating polynomial \n");
return 0;
}

```

```
}
```

Result:

```
C:\Users\Asha\Desktop\Untitled1.exe

Enter data : 1001100011
-----Enter the generating polynomial data1100
11
-----
Modified data is : 100110001100000
-----
Checksum is : 11111
-----
Transmitting codeword is : 100110001111111
-----
Enter received message
```

Fig. 1 Result of CRC- CCIT

EXPERIMENT NO 5

Aim : Implementation of Stop and Wait Protocol and Sliding Window Protocol.

Theory: Stop and wait and sliding window protocols are the data link layer protocols. Data link layer protocols provide flow and error control.

In stop and wait protocol, at the source when the frames are sent, a timer is set and the source waits for the acknowledgement from the destination. The source expects the acknowledgement from the receiver, before the time out. If the acknowledgement is successfully received, the next frame which is in queue at the source is sent, else the transmission is aborted.

In sliding window protocol, instead of sending the frames one by one, multiple frames are sent at once .Go Back N ARQ and Selective repeat protocols are the examples of sliding window protocols.

At the source, Timers, Acknowledgement and sequence numbers are used to send the frames to the destination.

Sequence numbers are used to ensure the correct reception of the frames and also frame numbers avoid the duplication of the packets at the receiver.

Program:

```
#include<stdio.h>
#include<unistd.h>

int sender();
int receiver();

int flag=0;

int main ()
{
    int i,n;
    printf("Enter the number  of frames you want to transmit\t");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        sender();
    }
    printf("All the %d frames are transmitted successfully\n",n);
    return(1);
}

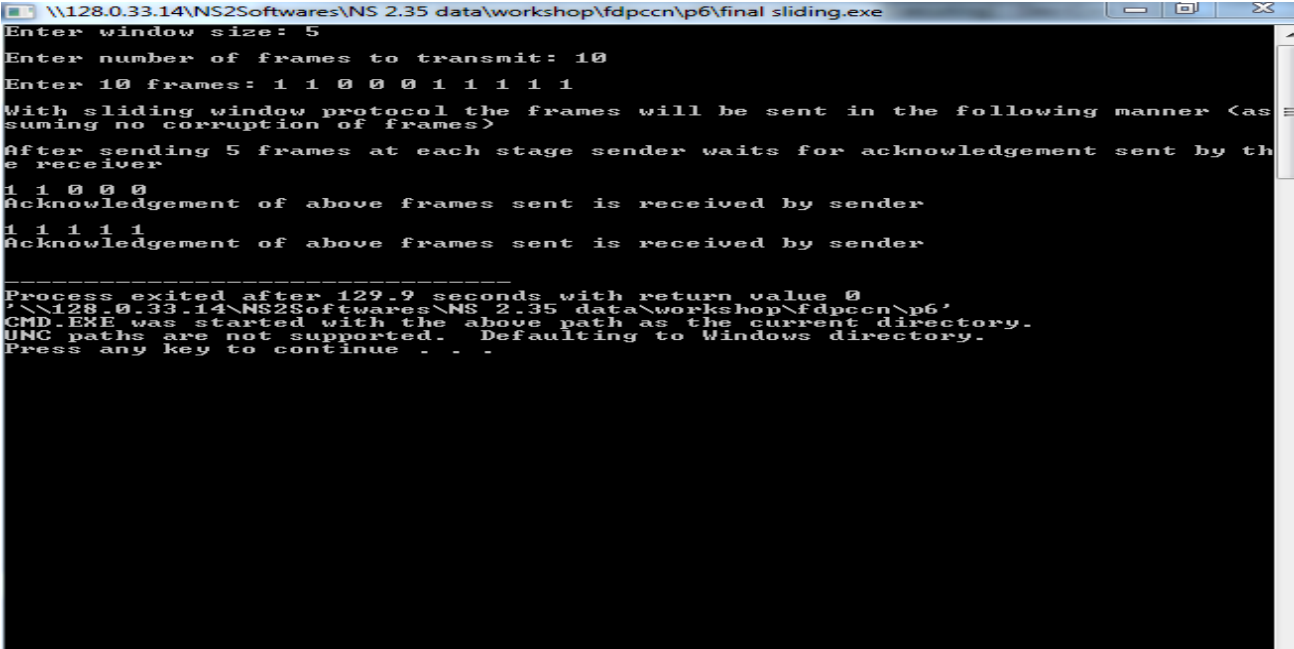
int sender()
{
    int ack;
```

```
    printf("Entered the sender function\n");
xyz:if (flag==0)
{
    printf("Save the frame\t");
    printf("transmit the 0th frame\t");
    printf("Start the timer\n");
    sleep(1);
    ack=receiver();
}
else
{
    printf("Save the frame\t");
    printf("transmit the 1st frame\t");
    printf("Start the timer\n");
    sleep(1);
    ack=receiver();
}
if(ack==0)
{
    printf("correct ack received,Frame is transmitted successfully stop the timer\n");
    return(1);
}
else if (ack==1)
{
    printf("Time out ouccerd\n");
    printf("retransmit the frame\n");
    goto xyz;
}
return 0;
}

int receiver()
{
    int ack,num;
    printf("Enter the number between 0 and 3 for acknowledgment\n");
    printf("0 : Frame received correctly\t");
    printf("1 : Frame is curropted\t");
    printf("2 : duplicate frame\t");
    printf("3 : No event\t");
    scanf("%d",&num);
    if (num==0)
    {
        printf("Frame if received correctly \n");
        if ( flag==0 )
            flag=1;
        else
            flag=0;
        ack=0;
        return(ack);
    }
    else if (num==1)
    {
        printf(" Frame is Corrupted Discard the frame \n");
        ack=1;
    }
}
```

```
        return(ack);
    }
    else if (num==2)
    {
        printf("Duplicate Frame is received \n");
        ack=0;
        return(ack);
    }
    else if (num==3)
    {
        printf("No event is occurred\n");
        ack=1;
        return(ack);
    }
}
```

Result:



```
\\128.0.33.14\NS2Softwares\NS 2.35 data\workshop\fdpccn\p6\final sliding.exe
Enter window size: 5
Enter number of frames to transmit: 10
Enter 10 frames: 1 1 0 0 0 1 1 1 1 1
With sliding window protocol the frames will be sent in the following manner (as
suming no corruption of frames)
After sending 5 frames at each stage sender waits for acknowledgement sent by th
e receiver
1 1 0 0 0
Acknowledgement of above frames sent is received by sender
1 1 1 1 1
Acknowledgement of above frames sent is received by sender

-----
Process exited after 129.9 seconds with return value 0
'\\128.0.33.14\NS2Softwares\NS 2.35 data\workshop\fdpccn\p6'
CMD.EXE was started with the above path as the current directory.
UNC paths are not supported. Defaulting to Windows directory.
Press any key to continue . . .
```

Fig. 1 Result of stop & wait ARQ

EXPERIMENT NO 5A

Theory: Sliding Window Protocol

In this protocol (and the next), the sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. In other words, the sender and receiver need to deal with only part of the possible sequence numbers. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window. We discuss both here. The send window is an imaginary box covering the sequence numbers of the data frames which can be in transit. In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent. The maximum size of the window is $2^m - 1$ for reasons that we discuss later. In this chapter, we let the size be fixed and set to the maximum value, but we will see in future chapters that some protocols may have a variable window size.

Figure 11.12 Send window for Go-Back-NARQ

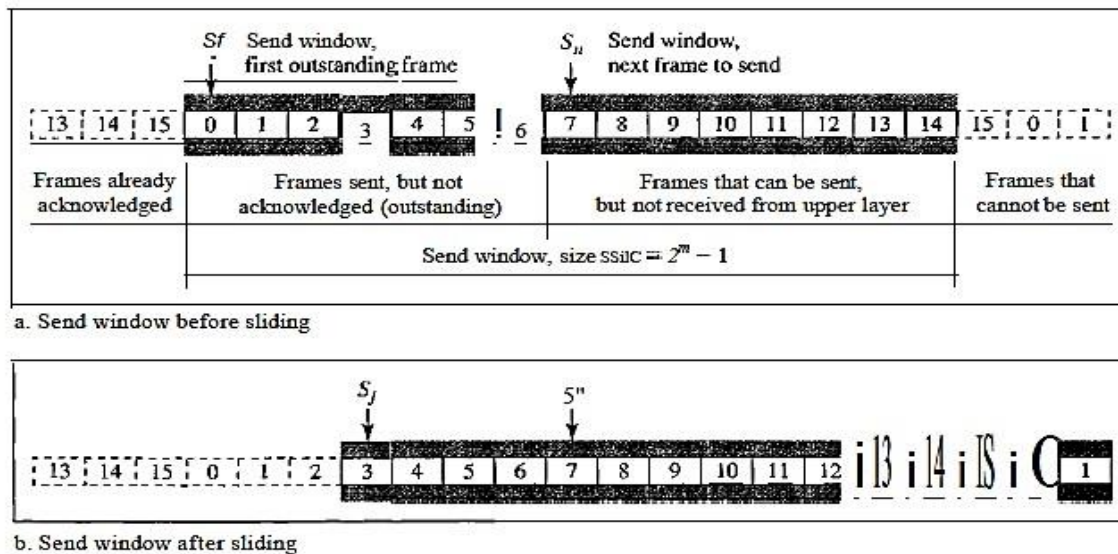
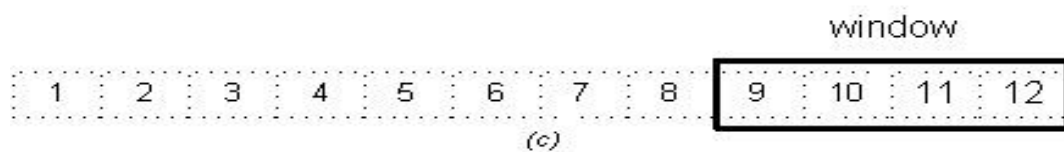
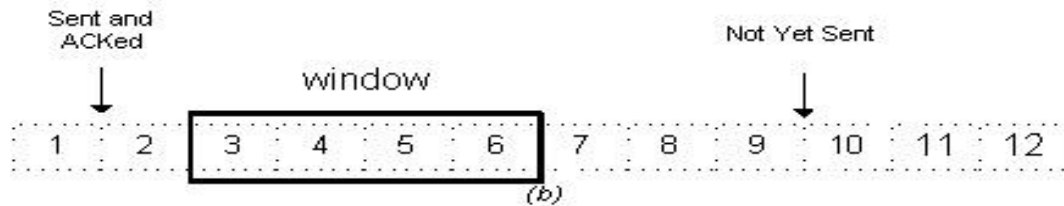
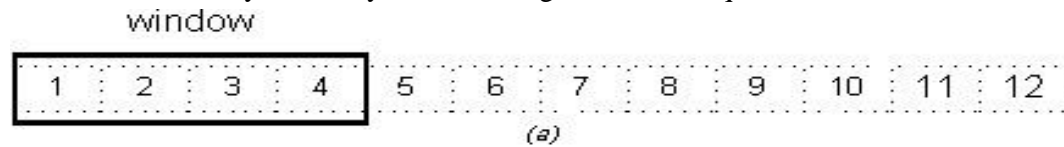


Figure 11.12 shows a sliding window of size 15 ($m = 4$). The window at any time divides the possible sequence numbers into four regions.

The first region, from the far left to the left wall of the window, defines the sequence numbers belonging to frames that are already acknowledged. The sender does not worry about these frames and keeps no copies of them.

The second region, colored in Figure 11.12a, defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these outstanding frames.

The third range, white in the figure, defines the range of sequence numbers for frames that can be sent; however, the corresponding data packets have not yet been received from the network layer. Finally, the fourth region defines sequence numbers that cannot



be used until the window slides. In Networking, Window simply means a buffer which has data frames that need to be transmitted.

Both sender and receiver agree on some window size. If window size = w then after sending w frames sender waits for the acknowledgement (ack) of the first frame. As soon as sender receives the acknowledgement of a frame it is replaced by the next frames to be transmitted by the sender. If receiver sends a collective or cumulative acknowledgement to sender then it understands that more than one frame is properly received, for e.g.:- if ack of frame 3 is received it understands that frame 1 and frame 2 are received properly.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define RTT 5
int main()
{

int window_size,i,f,frames[50];
printf("Enter window size: ");
scanf("%d",&window_size);
printf("\nEnter number of frames to transmit: ");
scanf("%d",&f);
printf("\nEnter %d frames: ",f);
for(i=1;i<=f;i++)

scanf("%d",&frames[i]);
printf("\nAfter sending %d frames at each stage sender waits for ACK ",window_size);
printf("\nSending frames in the following manner....\n\n");
```

```
for(i=1;i<=f;i++)
{
if(i%window_size!=0)
{
printf(" %d",frames[i]);
}
else
{
printf(" %d\n",frames[i]);
printf("SENDER: waiting for ACK...\n\n");
sleep(RTT/2);
printf("RECEIVER: Frames Received, ACK Sent\n");
printf("-----\n");
sleep(RTT/2);
printf("SENDER:ACK received, sending next frames\n");
}
}
if(f%window_size!=0)
{
printf("\nSENDER: waiting for ACK...\n");
sleep(RTT/2);
printf("\nRECEIVER:Frames Received, ACK Sent\n");
printf("-----\n");
sleep(RTT/2);
printf("SENDER:ACK received.");
}
return 0;
}
```

Result:

Both the protocols are implemented and executed and the result is displayed on the screen.

EXPERIMENT NO 6

Aim: Write a program for congestion control using leaky bucket algorithm.

Theory:

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.

Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called traffic shaping.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulated by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.

Program:

```

#include<stdio.h>
#include<stdlib.h>
#define MIN(x,y) (x>y)?y:x
int main()
{
    intorate,drop=0,cap,x,count=0,inp[10]={0},i=0,nsec,ch;
    printf("\n enter bucket size : ");
    scanf("%d",&cap);
    printf("\n enter output rate :");
    scanf("%d",&orate);
    do
    {
        printf("\n enter number of packets coming at second %d :",i+1);
        scanf("%d",&inp[i]);
        i++;
        printf("\n enter 1 to continue or 0 to quit.....");
        scanf("%d",&ch);
    }
    while(ch);
    nsec=i;
    printf("\n second \t recieved \t sent \t dropped \t remained \n");
    for(i=0;count || i<nsec;i++)
    {
        printf("%d",i+1);
        printf(" \t%d\t",inp[i]);
        printf(" \t %d\t",MIN((inp[i]+count),orate));
        if((x=inp[i]+count-orate)>0)
        {
            if(x>cap)
            {
                count=cap;
                drop=x-cap;
            }
            else
            {
                count=x;
                drop=0;
            }
        }
        else
        {
            drop=0;
            count=0;
        }
        printf(" \t %d \t %d \n",drop,count);
    }
    return 0;
}

```


Result:

```

\\128.0.33.14\\NS2Softwares\\NS 2.35 data\\workshop\\fdpccn\\p2\\final buck.exe
enter bucket size : 5
enter output rate :2
enter number of packets coming at second 1 :3
enter 1 to continue or 0 to quit.....1
enter number of packets coming at second 2 :5
enter 1 to continue or 0 to quit.....1
enter number of packets coming at second 3 :5
enter 1 to continue or 0 to quit.....0

second      recieved      sent      dropped      remained
1           3           2           0           1
2           5           2           0           4
3           5           2           2           5
4           0           2           0           3
5           0           2           0           1
6           0           1           0           0

-----
Process exited after 18.8 seconds with return value 0
'\\128.0.33.14\\NS2Softwares\\NS 2.35 data\\workshop\\fdpccn\\p2'
CMD.EXE was started with the above path as the current directory.
UNC paths are not supported.  Defaulting to Windows directory.
Press any key to continue . . .

```

Fig. 1 Result of leaky bucket / congestion control algorithm

VIVA QUESTIONS

1. What are functions of different layers?
2. Differentiate between TCP/IP Layers and OSI Layers
3. Why header is required?
4. What is the use of adding header and trailer to frames?
5. What is encapsulation?
6. Why fragmentation requires?
7. What is MTU?
8. Which layer imposes MTU?
9. Differentiate between flow control and congestion control.
10. Differentiate between Point-to-Point Connection and End-to-End connections.
11. What are protocols running in different layers?
12. What is Protocol Stack?
13. Differentiate between TCP and UDP.
14. Differentiate between Connectionless and connection oriented connection.
15. Why frame sorting is required?
16. What is meant by subnet?
17. What is meant by Gateway?
18. What is an IP address?
19. What is MAC address?
20. Why IP address is required when we have MAC address?
21. What is meant by port?
22. What are ephemeral port number and well known port numbers?
23. What is a socket?
24. What are the parameters of socket()?
25. Describe bind(), listen(), accept(), connect(), send() and recv().
26. What are system calls? Mention few of them.
27. What is IPC? Name three techniques.
28. Explain mkfifo(), open(), close() with parameters.
29. What is meant by file descriptor?
30. What is meant by traffic shaping?
31. How do you classify congestion control algorithms?
32. Differentiate between Leaky bucket and Token bucket.
33. How do you implement Leaky bucket?
34. How do you generate busty traffic?
35. What is the polynomial used in CRC-CCITT?
36. What are the other error detection algorithms?
37. What is difference between CRC and Hamming code?
38. Why Hamming code is called 7,4 code?
39. What is odd parity and even parity?
40. What is meant by syndrome?
41. What is generator matrix?
42. What are Routing algorithms?

43. How do you classify routing algorithms? Give examples for each.
44. What are drawbacks in distance vector algorithm?
45. How routers update distances to each of its neighbor?
46. How do you overcome count to infinity problem?
47. What is cryptography?
48. How do you classify cryptographic algorithms?
49. What is public key?
50. What is private key?
51. What are key cipher text and plaintext?
52. What is simulation?
53. What are advantages of simulation?
54. Differentiate between Simulation and Emulation.
55. What is meant by router?
56. What is meant by bridge?
57. What is meant by switch?
58. What is meant by hub?
59. Differentiate between route, bridge, switch and hub.
60. What is ping and telnet?
61. What is FTP?
62. What is BER?
63. What is meant by congestion window?
64. What is BSS?
65. What is incoming throughput and outgoing throughput?
66. What is collision?
67. How do you generate multiple traffics across different sender-receiver pairs?
68. How do you setup Ethernet LAN?
69. What is meant by mobile host?
70. What is meant by NCTUns?
71. What are dispatcher, coordinator and nctunsclient?
72. Name few other Network simulators
73. Differentiate between logical and physical address.
74. Which address gets affected if a system moves from one place to another place?
75. What is ICMP? What are uses of ICMP? Name few.
76. Which layer implements security for data?
77. What is connectionless and connection oriented protocol?
78. What is Contention?
79. What is Congestion window?
80. What is flow control?