VISVESVARAYA TECHNOLOGICAL UNIVERSITY,
Jnana Sangama, Belagavi – 590018

KLS VISHWNATHRAO DESHPANDE INSTITUTE OF TECHNOLOGY,
Haliyal – 581 329, Uttara Kannada

LABORATORY MANUAL

Course Title	:	COMPUTER NETWORKS & PROTOCOLS
Course Code	:	BEC702
Year / Semester	:	4th Year / 7thSem
Academic Year	:	2025-26
Course In-Charge	:	Prof. Anita M. H Prof. Raghavendra N

Department of Electronics and Communication Engineering

Signature of the Faculty with Date

HoD

KLS Vishwanathrao Deshpande Institute of Technology

(Accredited by NAAC with "A" Grade)

(Approved by AICTE, New Delhi, Affiliated to VTU, Belagavi)

(Recognized Under Section 2(f) by UGC, New Delhi)

Udyog Vidya Nagar, Haliyal - 581 329, Dist.: Uttara Kannada

www.klsvdit.edu.in | principal@klsvdit.edu.in | hodece@klsvdit.edu.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

College Vision and Mission Statements

Vision

To nurture talent & enrich society through excellence in technical education, research & innovation.

Mission

1. To augment innovative pedagogy & kindle quest for interdisciplinary learning & to enhance conceptual understanding.
2. To build competence, professional ethics & develop entrepreneurial thinking.
3. To strengthen industry institute partnership &explore global collaborations.
4. To inculcate culture of socially responsible citizenship.
5. To focus on holistic &sustainable development.

Department Vision,Mission,PEOs and PSOs Statements

Vision

To bring out talented, skilled, and sustainable Electronics and Communication Engineering Graduates through strong domain expertise to serve the Society with greater Professional Ethics.

Mission

1. To create and impart an active learning ambience to accomplish a high degree of Professional competencies
2. To inculcate innovative research and developmental thinking in effective Teaching and Learning processes for solving Societal challenges
3. To deliver the needs and requirements of the latest state of art of the Industry through quality multidisciplinary internship and training programs

PEOs

PEO 1:	To be successful in professional career in electronics, communication and allied industries by acquiring the knowledge in the fundamentals of Electronics and Communication Engineering principles and professional skills.
PEO 2:	To be in a position to analyze real life problems and design socially accepted and economically feasible solutions in the respective fields.
PEO 3:	To exhibit good communication skills in their professional career, lead a team with good leadership traits and good interpersonal relationship with the members related to other engineering streams.
PEO 4:	To involve themselves in lifelong learning and professional development by pursuing higher education and participation in research and development activities.
PEO 5:	To demonstrate professional and ethical responsibilities towards their profession, society and the environment.

PSOs

PSO 1:	An ability to use appropriate modern techniques for analysis, design and development of VLSI and Embedded Systems.
PSO 2:	Understand the architectural specifications of a communication system and determine their performance.

KLS Vishwanathrao Deshpande Institute of Technology

(Accredited by NAAC with "A" Grade)

(Approved by AICTE, New Delhi, Affiliated to VTU, Belagavi)

(Recognized Under Section 2(f) by UGC, New Delhi)

Udyog Vidya Nagar, Haliyal - 581 329, Dist.: Uttara Kannada

www.klsvdit.edu.in | principal@klsvdit.edu.in | hodece@klsvdit.edu.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Course Title	:	**COMPUTER NETWORKS & PROTOCOLS**
Course Code	:	**BEC702**
Year / Semester	:	**4th Year / 7th Semester**
Academic Year	:	**2025 – 2026**

Syllabus :

CLOs :

- Understand the layering architecture of OSI reference model and TCP/IP protocol suite.
- Understand the protocols associated with each layer.
- Learn the different networking architectures and their representations.
- Learn the various routing techniques and the transport layer services.

Cos :

CO1:	Understand the concepts of networking thoroughly.
CO2:	Identify the protocols and services of different layers.
CO3:	Distinguish the basic network configurations and standards associated with each network.
CO4:	Discuss and analyze the various applications that can be implemented on networks.

CO / PO Mapping :

COs	CO1	CO2	CO3	CO4
POs / PSOs	2, 3, 5, 12, PSO2	2, 3, 5, 12, PSO2	2, 3, 5, 12, PSO2	2, 3, 5,12,PSO2

Course In-Charge : **Prof. Anita M. H, Prof. Raghavendra N**

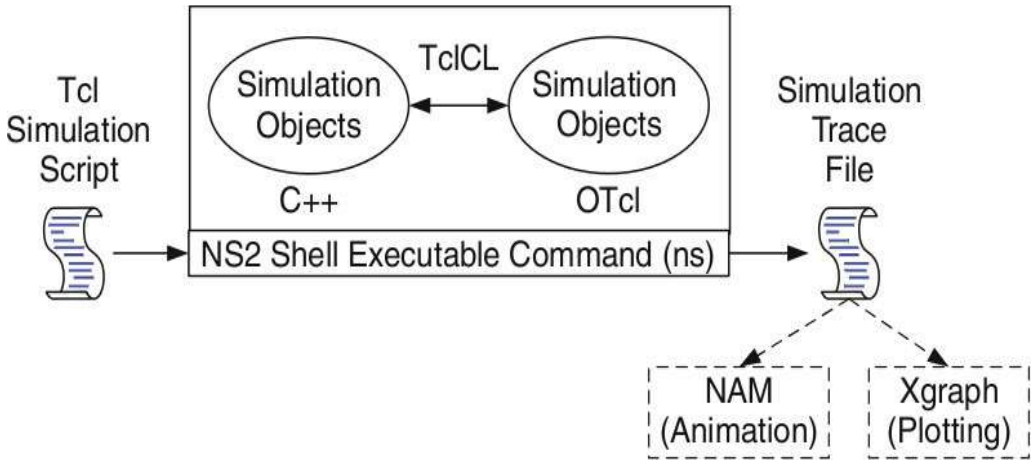
CO-PO Mapping Matrix

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
1		2	2		2							2		2
2		2	2		2							2		2
3		2	2		2							2		2
4		2	2		2							2		2

Introduction to NS-2

- Widely known as NS2, is simply an event driven simulation tool.
- Useful in studying the dynamic nature of communication networks.
- Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.
- In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors.

Basic Architecture of NS2



Tcl scripting

- Tcl is a general purpose scripting language. [Interpreter]
- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL

Syntax: command arg1 arg2 arg3

- **Hello World!**

puts stdout{Hello, World!}

Hello, World!

- **Simple Arithmetic**

  expr 7.2 / 4

- **Procedures**

```
proc Diag {a b} {
set c [expr sqrt($a * $a + $b * $b)]
        return $c }
puts ―Diagonal of a 3, 4 right triangle is [Diag 3 4]‖
```

Output: Diagonal of a 3, 4 right triangle is 5.0

- **Loops**

```
while{$i < $n}          for {set i 0} {$i < $n} {incr i}
{                       {
   ------                  ------
 }                       }
```

Wired TCL Script Components

Create the event scheduler

Open new files & turn on the tracing

Create the nodes

Setup the links

Configure the traffic type (e.g., TCP, UDP, etc)

Set the time of traffic generation (e.g., CBR, FTP)

Terminate the simulation

NS Simulator Preliminaries.

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

set ns [new Simulator]

Which is thus the first line in the tcl script? This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because

it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the inst allation of the class Simulator using the reserved word new.

In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using ―open‖ command:

#Open the Trace file

```
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
```

#Open the NAM trace file

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

The above creates a dta trace file called ―out.tr‖ and a nam visualization trace file called ―out.nam‖.Within the tcl script,these files are not called explicitly by their names,but instead by pointers that are declared above and called ―tracefile1‖ and ―namfile‖ respectively.Remark that they begins with a # symbol.The second line open the file ―out.tr‖ to be used for writing,declared with the letter ―w‖.The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

The last line tells the simulator to record all simulation traces in NAM input format.It also gives the file name that the trace will be written to later by the command $ns flush- trace.In our case,this will be the file pointed at by the pointer ―$namfile‖,i.e the file ―out.tr‖.

The termination of the program is done using a ―finish‖ procedure.

#Define a „finish‟ procedure

```
Proc finish {} {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close $namfile
Exec nam out.nam &
Exit 0
}
```

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method **—flush-trace"** will dump the traces on the respective files. The tcl command **—close"** closes the trace files defined before and **exec** executes the nam program for visualization. The command **exit** will ends the application and return the number 0 as status to the system. Zero is the default for a clean exit. Other values can be used to say that is a exit because something fails. At the end of ns program we should call the procedure —finish‖ and specify at what time the termination should occur. For example, will be used to call **—finish‖** at time 125sec.Indeed,the **at** method of the simulator allows us to schedule events explicitly.

```
$ns at 125.0 "finish"
```

The simulation can then begin using the command

```
$ns run
```

Definition of a network of links and nodes

The way to define a node is

```
set n0 [$ns node]
```

The node is createdwhich is printed by the variable n0. When we shall refer to that node in the script we shall thus write $n0.

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

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

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

To define a directional link instead of a bi-directional one, we should replace —duplex- link‖ by —simplex-link‖.

In NS, an output queue of a node is implemented as a part of each link whose input is that node. The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Discard)

In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

```
#set Queue Size of link (n0-n2) to 20

$ns queue-limit $n0 $n2 20
```

Agents and Applications

We need to define routing (sources, destinations) the agents (protocols) the application that use them.

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

```
set tcp [new Agent/TCP]
```

The command **$ns attach-agent $n0 $tcp** defines the source node of the tcp connection.

The command

```
set sink [new Agent /TCPSink]
```

Defines the behavior of the destination node of TCP and assigns to it a pointer called sink.

#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 packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false
```

Above shows the definition of a CBR application using a UDP agent

The command **$ns attach-agent $n4 $sink** defines the destination node. The command **$ns connect $tcp $sink** finally makes the TCP connection between the source and destination nodes.

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command **$tcp set packetSize_ 552**.

When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **$tcp set fid_ 1** that assigns to the TCP connection a flow identification of ―1‖.We shall later give the flow identification of ―2‖ to the UDP connection.

CBR over UDP

A UDP source and destination is defined in a similar way as in the case of TCP.

Instead of defining the rate in the command $cbr set rate_ 0.01Mb, one can define the time interval between transmission of packets using the command.

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

```
$cbr set packetSize_ <packet size>
```

Scheduling Events

NS is a discrete event based simulation. The tcp script defines when event should occur. The initializing command set ns [new Simulator] creates an event scheduler, and events are then scheduled using the format:

```
$ns at <time><event>
```

The scheduler is started when running ns that is through the command $ns run.

The beginning and end of the FTP and CBR application can be done through the following command

```
$ns at 0.1 "$cbr start"

$ns at 1.0 " $ftp start"

$ns at 124.0 "$ftp stop"

$ns at 124.5 "$cbr stop"
```

Structure of Trace Files

When tracing into an output ASCII file, the trace is organized in 12 fields as follows in fig shown below, The meaning of the fields are:

Event	Time	From Node	To Node	PKT Type	PKT Size	Flags	Fid	Src Addr	Dest Addr	Seq Num	Pkt Id

1. The first field is the event type. It is given by one of four possible symbols r, +, -, d which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.
2. The second field gives the time at which the event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.
9. This is the source address given in the form of ―node.port‖.
10. This is the destination address, given in the same form.
11. This is the network layer protocol‘s packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes
12. The last field shows the Unique id of the packet.

XGRAPH

The xgraph program draws a graph on an x-display given data read from either data file or from standard input if no files are specified. It can display upto 64 independent data sets using different colors and line styles for each set. It annotates the graph with a title, axis labels, grid lines or tick marks, grid labels and a legend.

Syntax:

```
Xgraph [options] file-name
```

Options are listed here

/-bd <color> (Border)

This specifies the border color of the xgraph window.

/-bg <color> (Background)

This specifies the background color of the xgraph window.

/-fg<color> (Foreground)

This specifies the foreground color of the xgraph window.

/-lf <fontname> (LabelFont)

All axis labels and grid labels are drawn using this font.

/-t<string> (Title Text)

This string is centered at the top of the graph.

/-x <unit name> (XunitText)

This is the unit name for the x-axis. Its default is ―X‖.

/-y <unit name> (YunitText)

This is the unit name for the y-axis. Its default is ―Y‖.

Awk- An Advanced

awk is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers.

awk is not just a command, but a programming language too. In other words, awk utility is a pattern scanning and processing language. It searches one or more files to see if they contain lines that match specified patterns and then perform associated actions, such as writing the line to the standard output or incrementing a counter each time it finds a match.

Syntax:

```
awk option 'selection_criteria {action}' file(s)
```

Here, selection_criteria filters input and select lines for the action component to act upon. The selection_criteria is enclosed within single quotes and the action within the curly braces. Both the selection_criteria and action forms an awk program.

Example: $ awk „/manager/ {print}‟ emp.lst

Variables

Awk allows the user to use variables of there choice. You can now print a serial number, using the variable kount, and apply it those directors drawing a salary exceeding 6700:

```
$ awk –F"|" „$3 == "director" && $6 > 6700
{ kount =kount+1
printf " %3f %20s %-12s %d\n", kount,$2,$3,$6 }‟ empn.lst
```

THE –f OPTION: STORING awk PROGRAMS IN A FILE

You should holds large awk programs in separate file and provide them with the awk extension for easier identification. Let‘s first store the previous program in the file empawk.awk:

$ cat empawk.awk

Observe that this time we haven‘t used quotes to enclose the awk program. You can now use awk with the –f *filename* option to obtain the same output:

```
Awk –F"|" –f empawk.awk empn.lst
```

THE BEGIN AND END SECTIONS

Awk statements are usually applied to all lines selected by the address, and if there are no addresses, then they are applied to every line of input. But, if you have to print something

before processing the first line, for example, a heading, then the BEGIN section can be used gainfully. Similarly, the end section useful in printing some totals after processing is over.

The BEGIN and END sections are optional and take the form

BEGIN {action}

END {action}

These two sections, when present, are delimited by the body of the awk program. You can use them to print a suitable heading at the beginning and the average salary at the end.

BUILT-IN VARIABLES

Awk has several built-in variables. They are all assigned automatically, though it is also possible for a user to reassign some of them. You have already used NR, which signifies the record number of the current line. We'll now have a brief look at some of the other variable.

The FS Variable*:* as stated elsewhere, awk uses a contiguous string of spaces as the default field delimiter. FS redefines this field separator, which in the sample database happens to be the |. When used at all, it must occur in the BEGIN section so that the body of the program knows its value before it starts processing:

BEGIN {FS="|"}

This is an alternative to the –F option which does the same thing.

The OFS Variable*:* when you used the print statement with comma-separated arguments, each argument was separated from the other by a space. This is awk's default output field separator, and can reassigned using the variable OFS in the BEGIN section:

BEGIN { OFS="~" }

When you reassign this variable with a ~ (tilde), awk will use this character for delimiting the print arguments. This is a useful variable for creating lines with delimited fields.

The NF variable*:* NF comes in quite handy for cleaning up a database of lines that don't contain the right number of fields. By using it on a file, say emp.lst, you can locate those lines not having 6 fields, and which have crept in due to faulty data entry:

$awk „BEGIN {FS = "|"}

NF! =6 {

Print "Record No ", NR, "has", "fields"}" empx.lst

Experiment No: 1

FOUR NODE POINT TO POINT NETWORK

Objectives:

- To implement a point to point network with four nodes and duplex links between them.
- To analyze the network performance by setting the queue size and varying the bandwidth.

Theory: A Four Node Point-to-Point Network is a basic network topology involving four devices (or nodes), where connections are established directly between specific pairs of nodes, typically using dedicated links (point-to-point). This structure is often used to understand fundamental networking concepts, including routing, switching, and data transmission**.** Involves a direct link between two nodes. No intermediate devices like routers or switches are involved. Ensures dedicated bandwidth and low latency between connected nodes.

```
set ns [new Simulator]                          # Letter S is capital
set nf [open PA1.nam w]                         # open a nam trace file in write mode
$ns namtrace-all $nf                            # nf nam filename
set tf [open PA1.tr w]                          # tftrace filename
$ns trace-all $tf

proc finish { } {
        global ns nf tf
        $ns flush-trace                         # clears trace file contents
        close $nf
        close $tf
        exec nam PA1.nam &
        exit 0
}
set n0 [$ns node] # creating 4 nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 200Mb 10ms DropTail     # establishing links
$ns duplex-link $n1 $n2 200Mb 10ms DropTail
$ns duplex-link $n2 $n3 200Mb 10ms DropTail

$ns queue-limit $n0 $n2 10
$ns queue-limit $n1 $n2 10
# $ns queue-limit $n2 $n3 10

set udp0 [new Agent/UDP]                        # attaching transport layer protocols
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]          # attaching application layer protocols
$cbr0 set packetSize_500
$cbr0 set interval_0.005
$cbr0 attach-agent $udp0
```

```
set null0 [new Agent/Null]                    # creating sink(destination) node
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0

$ns at 0.1 "$cbr0 start"
$ns at 1.0 "finish"
$ns run
```

AWK file:(Open a new editor using “vi command” and write awk file and save with “.awk” extension)

```
#immediately after BEGIN should open braces „{„
BEGIN{c=0;}
{
if($1= ="d")
  {       c++;
          printf("%s\t%s\n",$5,$11);
   }
 }
END{printf("The number of packets dropped
=%d\n",c);}
```

Steps for execution

- Open vi editor and type program. Program name should have the extension “ .tcl ” ***[root@localhost ~]# vi lab1.tcl***
- Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press Enter key.
- Open vi editor and type awk program. Program name should have the extension “.awk ”

  [root@localhost ~]# vi lab1.awk
- Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press Enter key.
- Run the simulation program

  [root@localhost~]# ns lab1.tcl
- Here “ns” indicates network simulator. We get the topology shown in the snapshot.
- Now press the play button in the simulation window and the simulation will begins.

- After simulation is completed run awk file to see the output ,

  [root@localhost~]# awk –f lab1.awk lab1.tr
- To see the trace file contents open the file as ,

  [root@localhost~]# vi lab1.tr

Trace file contains 12 columns:

Event type, Event time, From Node, To Node, Packet Type, Packet Size, Flags (indicated by --------), Flow ID, Source address, Destination address, Sequence ID, Packet ID.

OUTPUT:

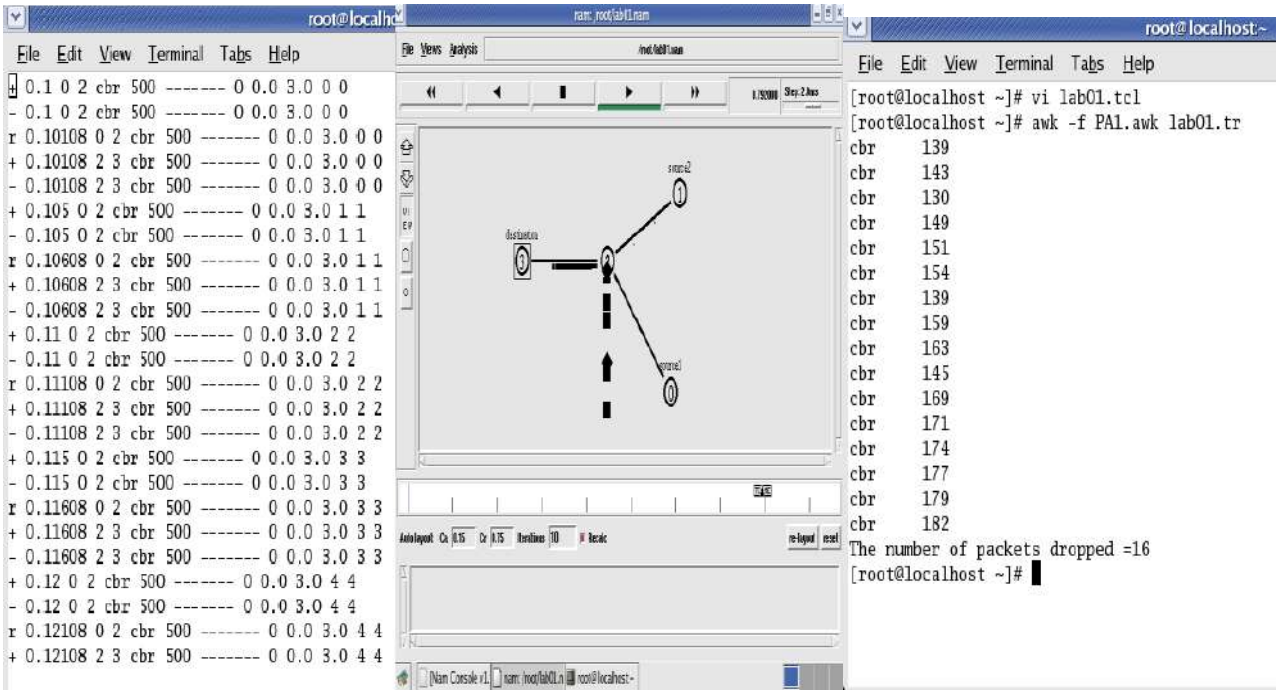


Fig. 1: Four node point to point network

Experiment No: 2

FOUR NODE POINT TO POINT NETWORK WITH TCP and UDP

Objectives:

- To 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.
- To apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets sent by TCP/UDP.

Theory: A Four Node Point-to-Point Network using TCP (Transmission Control Protocol**)** and UDP **(**User Datagram Protocol**)** demonstrates how reliable and unreliable data transmission can occur between multiple hosts connected by direct links. Let the nodes be: A, B, C, and D – each can act as a host or a router depending on the network configuration. Each node is connected to at least one other node via a point-to-point link**.** Communication paths are established between nodes using routing (static/dynamic).Depending on the topology, the network can be a ring, mesh, or star**.**

```
set ns [new Simulator] set
nf [open lab2.nam w]
$ns namtrace-all $nf set
tf [open lab2.tr w]
$ns trace-all $tf
proc finish { }
{
  global ns nf tf
  $ns flush-trace
  close $nf
  close $tf
  exec nam lab2.nam &
  exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n2 10Mb 1ms DropTail
$ns duplex-link $n1 $n2 10Mb 1ms DropTail
$ns duplex-link $n2 $n3 10Mb 1ms DropTail

set tcp0 [new Agent/TCP]                    # letters A,T,C,P are capital
$ns attach-agent $n0 $tcp0
set udp1 [new Agent/UDP]                    # letters A,U,D,P are capital
$ns attach-agent $n1 $udp1
set null0 [new Agent/Null]                  # letters A and N are capital
```

```
$ns attach-agent $n3 $null0
set sink0 [new Agent/TCPSink]                    # letters A,T,C,P,S are capital
$ns attach-agent $n3 $sink0  set
ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$ns connect $tcp0 $sink0
$ns connect $udp1 $null0
$ns at 0.1 "$cbr1 start"

$ns at 0.2 "$ftp0 start"

$ns at 0.5 "finish"
$ns run
```

AWK file:(Open a new editor using “vi command” and write awk file and save with “.awk” extension)

```
BEGIN

{
   udp=0;
   tcp=0;
}
{
  if($1= = “r” && $5 = = “cbr”)
   {
    udp++;
   }
 else if($1 = = “r” && $5 = = “tcp”)
  {
    tcp++;
   }
}
END
{
   printf(“Number of packets sent by TCP = %d\n”, tcp);
   printf(“Number of packets sent by UDP=%d\n”,udp);
}
```

Steps for execution:

- Open vi editor and type program. Program name should have the extension “ .tcl ”

  [root@localhost ~]# vi lab2.tcl
- Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press Enter key.
- Open vi editor and type awk program. Program name should have the extension “.awk ”

  [root@localhost ~]# vi lab2.awk
- Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press Enter key.
- Run the simulation program [root@localhost~]# ns lab2.tcl

Here “ns” indicates network simulator. We get the topology shown in the snapshot.
Now press the play button in the simulation window and the simulation will begins.

- After simulation is completed run awk file to see the output ,

  [root@localhost~]# awk –f lab2.awk lab2.tr
- To see the trace file contents open the file as ,

  [root@localhost~]# vi lab2.tr

OUTPUT:

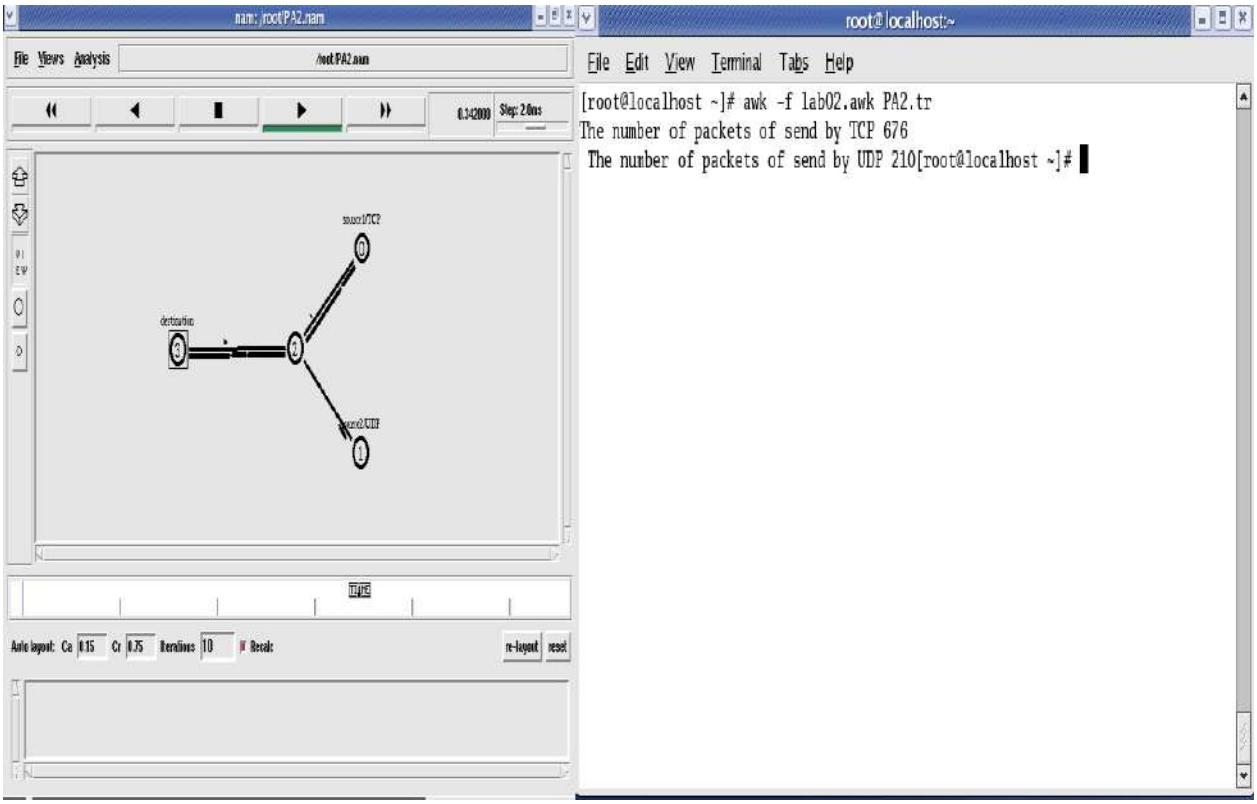


Fig. 2: Four node point to point network with tcp and ud

Experiment No: 3

ETHERNET LAN USING N-NODES

Objectives:

- To implement Ethernet LAN using n (6-10) nodes.
- To compare the throughput by changing the error rate and data rate.

Theory: An Ethernet LAN (Local Area Network) is a network where multiple computers (or nodes) are connected using Ethernet technology, allowing them to communicate, share resources, and **access** the internet. It operates over a small geographical area, such as a room, building, or campus. In an **N-**node Ethernet LAN**,** N devices (e.g., computers, printers, routers) are connected using Ethernet standards. Ethernet communication uses a frame-based structure**,** where data is encapsulated in frames that contain both source and destination MAC addresses, type fields, payload (data), and error-checking mechanisms such as CRC (Cyclic Redundancy Check). In older half-duplex Ethernet networks, CSMA/CD (Carrier Sense Multiple Access with Collision Detection**)** was used to avoid data collisions. However, modern full-duplex Ethernet switches have eliminated collisions by allowing simultaneous bidirectional communication between nodes.

```
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"
```

```
$ns make-lan "$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_ 0.1

# 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_ 0.0001
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
```

AWK file:(Open a new editor using “vi command” and write awk file and save with “.awk” extension)

```
BEGIN{
pkt=0;
time=0;
}
{
   if($1= ="r" && $3= ="9" && $4= ="7")
   {
     pkt = pkt + $6;
     time =$2;
   }
```

```
}
END
{
  printf("throughput:%fMbps",(( pkt / time) * (8 / 1000000)));
}
```

Steps for execution

- Open vi editor and type program. Program name should have the extension " .tcl "

  [root@localhost ~]# vi lab3.tcl

- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.
- Open vi editor and type awk program. Program name should have the extension ".awk "

  [root@localhost ~]# vi lab3.awk
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.
- Run the simulation program

  [root@localhost~]# ns lab3.tcl
  - Here "ns" indicates network simulator. We get the topology shown in the snapshot.
  - Now press the play button in the simulation window and the simulation will begins.
- After simulation is completed run awk file to see the output ,

  [root@localhost~]# awk –f lab3.awk lab3.tr
- To see the trace file contents open the file as ,

  [root@localhost~]# vi lab3.tr

Here **"h"** indicates host

OUTPUT:

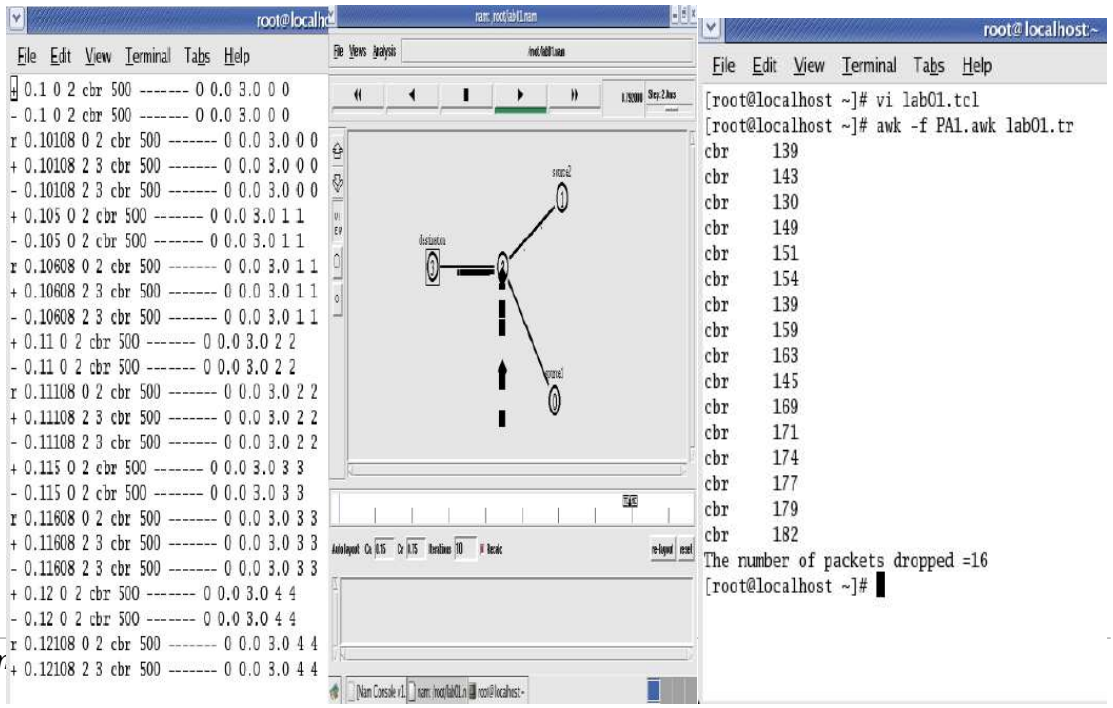


Fig. 3: Changing the error rate and data rate.

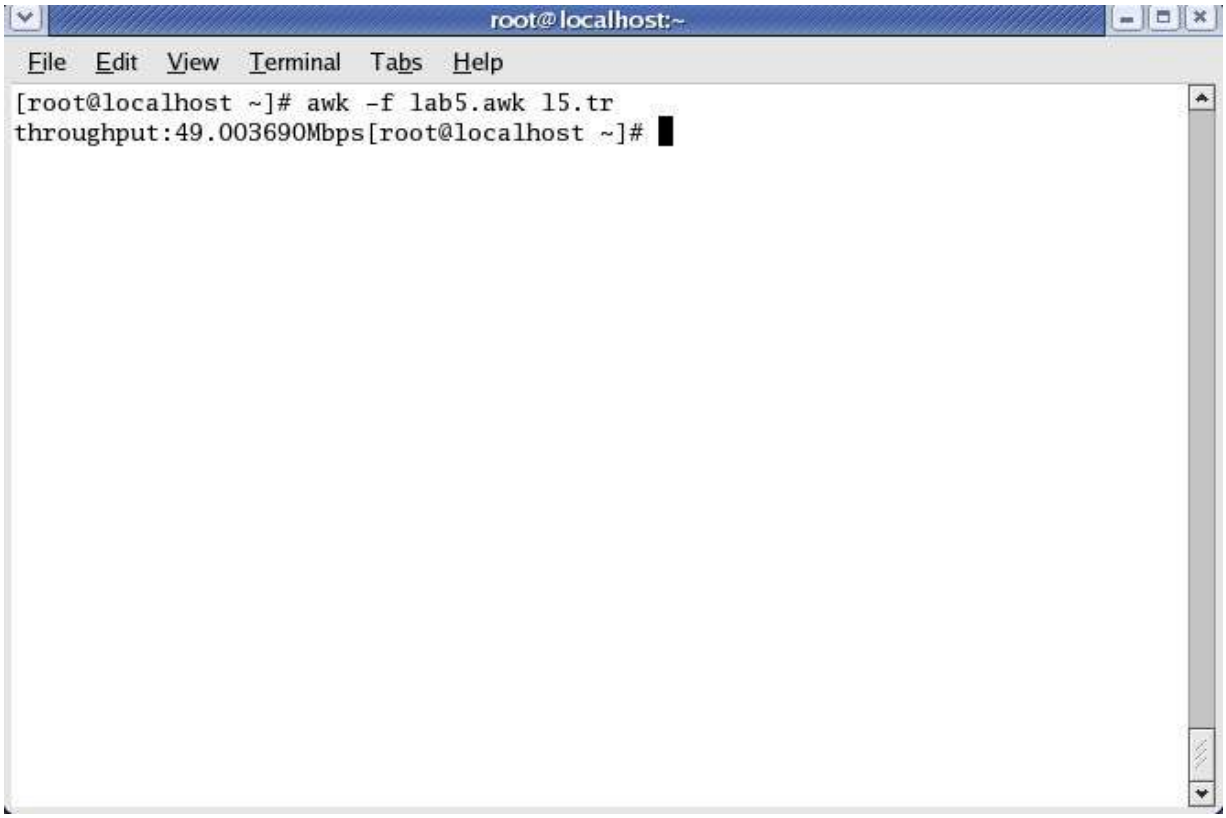


Fig. 4: Changing the error rate and data rate.

This above output is for error rate 0.1. During next execution of simulation change error rate to 0.3, 0.5,…..and check its effect on throughput.

Experiment No: 4

ETHERNET LAN USING N-NODES WITH MULTIPLE TRAFFIC

Objectives:

- To implement Ethernet LAN using n nodes and assign multiple traffic to the nodes
- T obtain congestion window for different sources/ destinations.

Theory: An Ethernet LAN (Local Area Network) using N-nodes with multiple traffic types refers to a network setup where several devices—such as computers, printers, IP phones, servers, and IoT devices —are interconnected and simultaneously generating or receiving various kinds of data traffic. This traffic can include file transfers, internet browsing, video streaming, voice-over-IP (VoIP), **live** conferencing, email communication, and real-time sensor data**,** all occurring over the same Ethernet infrastructure. Each node communicates over the network using Ethernet frames based on the IEEE 802.3 standard, and these frames are transmitted through network switches that manage and forward data based on the destination MAC addresses.In such a multi-traffic environment, the Ethernet switch plays a crucial **role** by isolating traffic between nodes and ensuring efficient delivery without unnecessary broadcasting. The switch maintains a MAC address table**,** which allows it to forward each frame only to the port corresponding to the destination node, improving bandwidth usage and reducing congestion.

```
set ns [new Simulator] set tf [open pgm4.tr w]
$ns trace-all $tf
set nf [open pgm4.nam w]
$ns namtrace-all $nf

set n0 [$ns node]
$n0 color "magenta"
$n0  label  "src1"
set n1  [$ns node]
set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2"
set n3 [$ns node]
$n3 color "blue"
$n3  label  "dest2"
set  n4  [$ns  node]
set n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"

$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/ DropTail Mac/802_3  # should
come in single line
$ns duplex-link $n4 $n5 1Mb 1ms DropTail

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

```
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5

$ns connect $tcp0 $sink5

set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp2 $sink3

set file1 [open file1.tr w]
$tcp0 attach $file1


set file2 [open file2.tr w]
$tcp2 attach $file2

$tcp0 trace cwnd_  # must put underscore ( _ ) after cwnd and no space between them
$tcp2 trace cwnd_

proc finish { }
{
  global ns nf tf
  $ns flush-trace
  close $tf
  close $nf
  exec nam pgm4.nam &
  exit 0
}


$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"
$ns at 16 "finish"
$ns run
```

AWK file: (Open a new editor using “vi command” and write awk file and save with “.awk” extension)

cwnd:- means congestion window

```
BEGIN {
}
{
   if($6= ="cwnd_")  # don‟t leave space after writing cwnd_ printf("%f\t%f\t\n",$1,$7); #
```

```
   you must put \n in printf
}
END {
}
```

Steps for execution

- Open vi editor and type program. Program name should have the extension " .tcl "

  [root@localhost ~]# vi lab4.tcl
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.
- Open vi editor and type awk program. Program name should have the extension ".awk "

  [root@localhost ~]# vi lab4.awk
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.
- Run the simulation program

  [root@localhost~]# ns lab4.tcl
- After simulation is completed run awk file to see the output ,

  [root@localhost~]# awk –f lab4.awk file1.tr >a1
  [root@localhost~]# awk –f lab4.awk file2.tr >a2
  [root@localhost~]# xgraph a1 a2
- Here we are using the congestion window trace files i.e. file1.tr and file2.tr and we are redirecting the contents of those files to new files say a1 and a2 using output redirection operator (>).
- To see the trace file contents open the file as ,

  [root@localhost~]# vi lab4.tr

Topology:

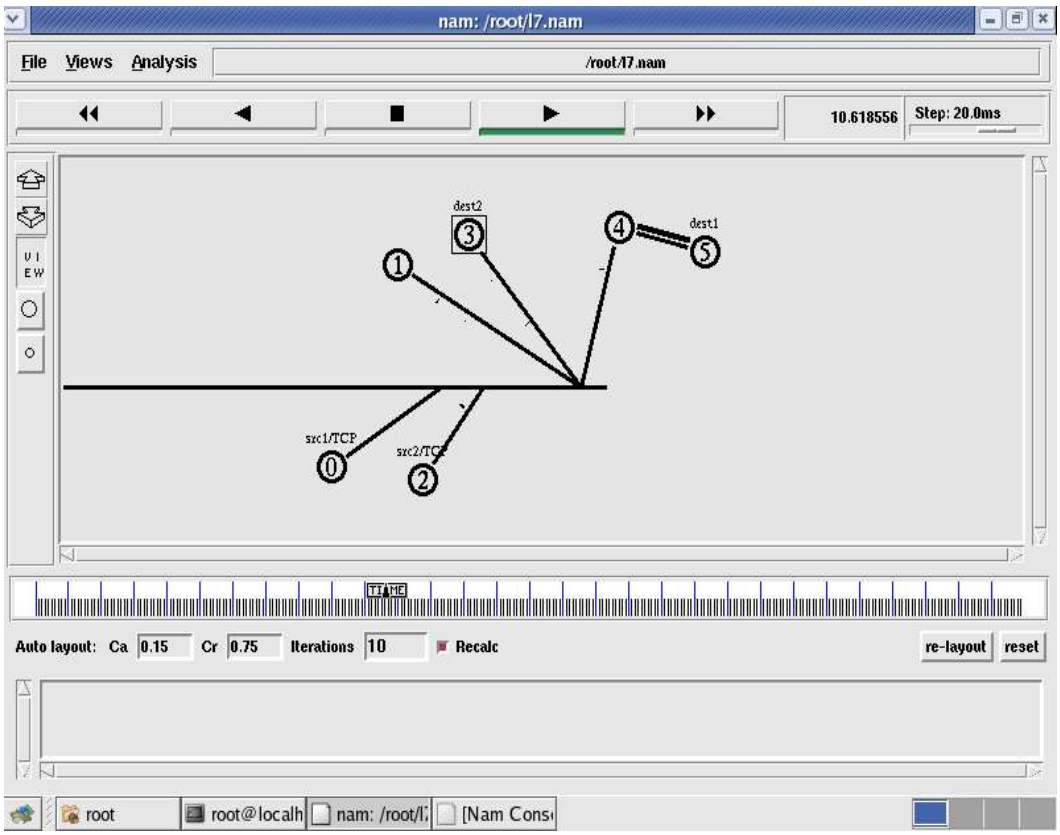


Fig. 4.1: Ethernet LAN using n-nodes with multiple traffic

OUTPUT:

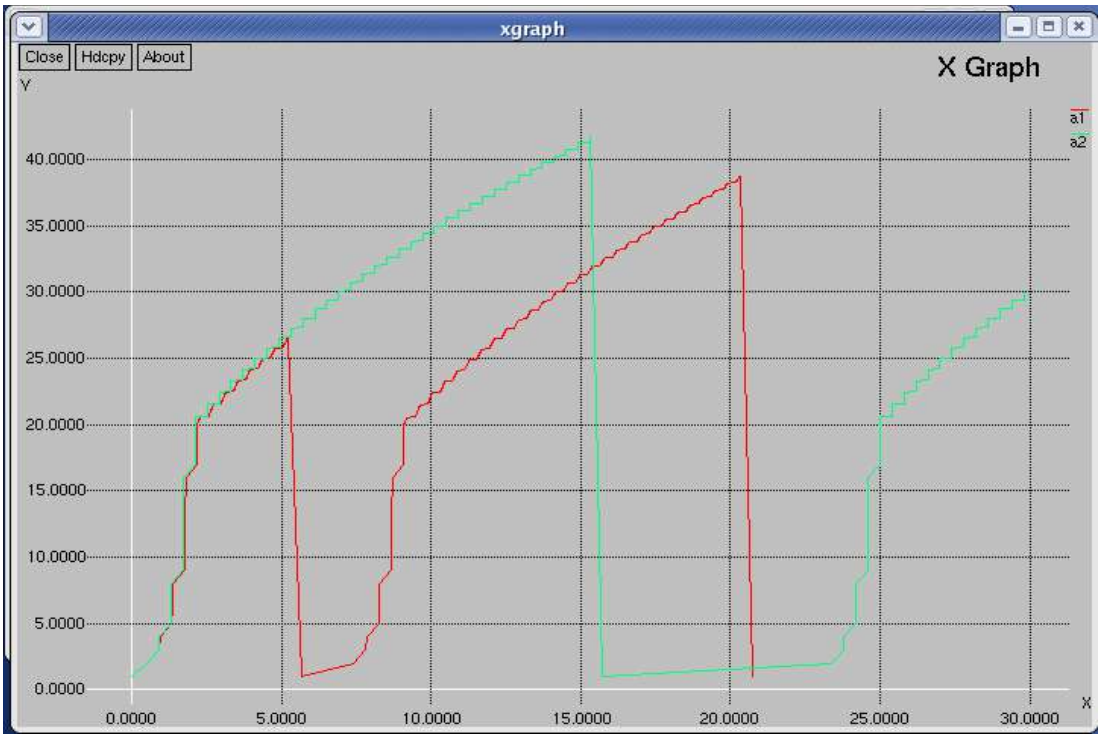


Fig. 4.2: X graph ethernet lAN using n-nodes with multiple traffic

Experiment No: 5

SIMPLE ESS WITH WIRELESS LAN

Objective:

- To Implement ESS with transmission nodes in Wireless LAN and obtain the performance parameters

Theory: A Simple Extended Service Set (ESS) in a Wireless LAN (WLAN) **is** a network configuration that connects multiple Basic Service Sets (BSSs**)** to provide broader wireless coverage and seamless mobility to wireless clients. Each BS**S** consists of a single Access Point (AP) and the wireless devices **(**stations) connected to it, forming a localized area of wireless communication. In an ESS**,** two or more access points are interconnected via a wired backbone—typically an Ethernet LAN—and are configured with the same SSID (Service Set Identifier)**.** This allows mobile clients to roam between different **access** point**s** while maintaining continuous network access without needing to reconnect manually. For example, in a school or office building, multiple APs can be deployed in different rooms or floors, forming an ESS that enables a user to walk from one area to another without losing connectivity.

```
set ns [new Simulator]
set tf [open lab5.tr w]
$ns trace-all $tf
set topo [new Topography]
$topo load_flatgrid 1000 1000 set
nf [open lab5.nam w]
$ns namtrace-all-wireless $nf 1000 1000

$ns node-config -adhocRouting DSDV \
                -llType LL \
                -macType Mac/802_11 \
                -ifqType Queue/DropTail \
                -ifqLen 50 \
                -phyType Phy/WirelessPhy \
                -channelType Channel/WirelessChannel \
                -propType Propagation/TwoRayGround \
                -antType Antenna/OmniAntenna \
                -topoInstance $topo \
                -agentTrace ON \
                -routerTrace ON

create-god 3
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"

$n0 set X_ 50
$n0 set Y_ 50
```

```
$n0 set Z_ 0
$n1 set X_ 100
$n1 set Y_ 100
$n1 set Z_ 0
$n2 set X_ 600
$n2 set Y_ 600
$n2 set Z_ 0

$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 100 100 25"
$ns at 0.1 "$n2 setdest 600 600 25" set
tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]


$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2
$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"

$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 70 70 15" proc
finish { } {
        global ns nf tf
        $ns flush-trace
        exec nam lab5.nam  &
        close $tf
        exit 0
}
$ns at 250 "finish"
$ns run
```

AWK file: (Open a new editor using “vi command” and write awk file and save with “.awk” extension)

```
BEGIN{
        count1=0
        count2=0
        pack1=0
        pack2=0
        time1=0
        time2=0
         }
{
        if($1= ="r"&& $3= ="_1_" && $4= ="AGT")
        {
```

```
                count1++ pack1=pack1+
                $8 time1=$2
        }

        if($1= ="r" && $3= ="_2_" && $4= ="AGT")
        {
                count2++
                pack2=pack2+$8
                time2=$2
        }
}

END{
printf("The Throughput from n0 to n1: %f Mbps \n", ((count1*pack1*8)/(time1*1000000)));

printf("The Throughput from n1 to n2: %f Mbps", ((count2*pack2*8)/(time2*1000000)));
}
```

Steps for execution

- Open vi editor and type program. Program name should have the extension " .tcl "

  [root@localhost ~]# vi lab5.tcl
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.
- Open vi editor and type awk program. Program name should have the extension ".awk "

  [root@localhost ~]# vi lab5.awk
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.
- Run the simulation program

  [root@localhost~]# ns lab5.tcl
  - Here "ns" indicates network simulator. We get the topology shown in the snapshot.
  - Now press the play button in the simulation window and the simulation will begins.
- After simulation is completed run awk file to see the output ,

  [root@localhost~]# awk –f lab5.awk lab5.tr
- To see the trace file contents open the file as ,

  [root@localhost~]# vi lab5.tr

OUTPUT:

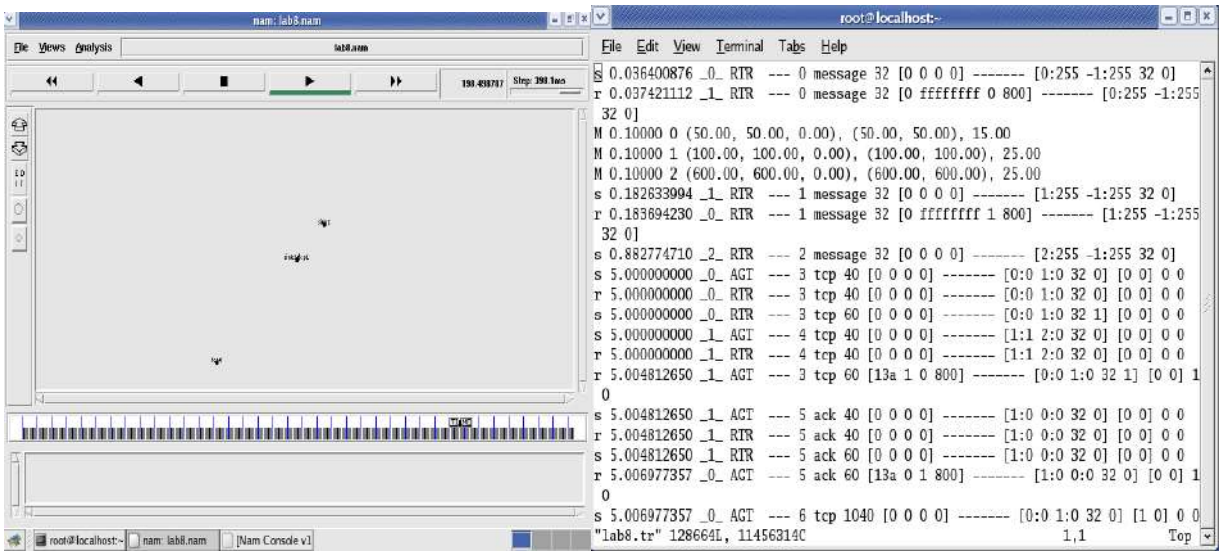


Fig. 5.1 Node 2 is coming back from node 3 towards node1 and Trace File

Here **"M"** indicates mobile nodes, **"AGT"** indicates Agent Trace, **"RTR"** indicates Route Trace

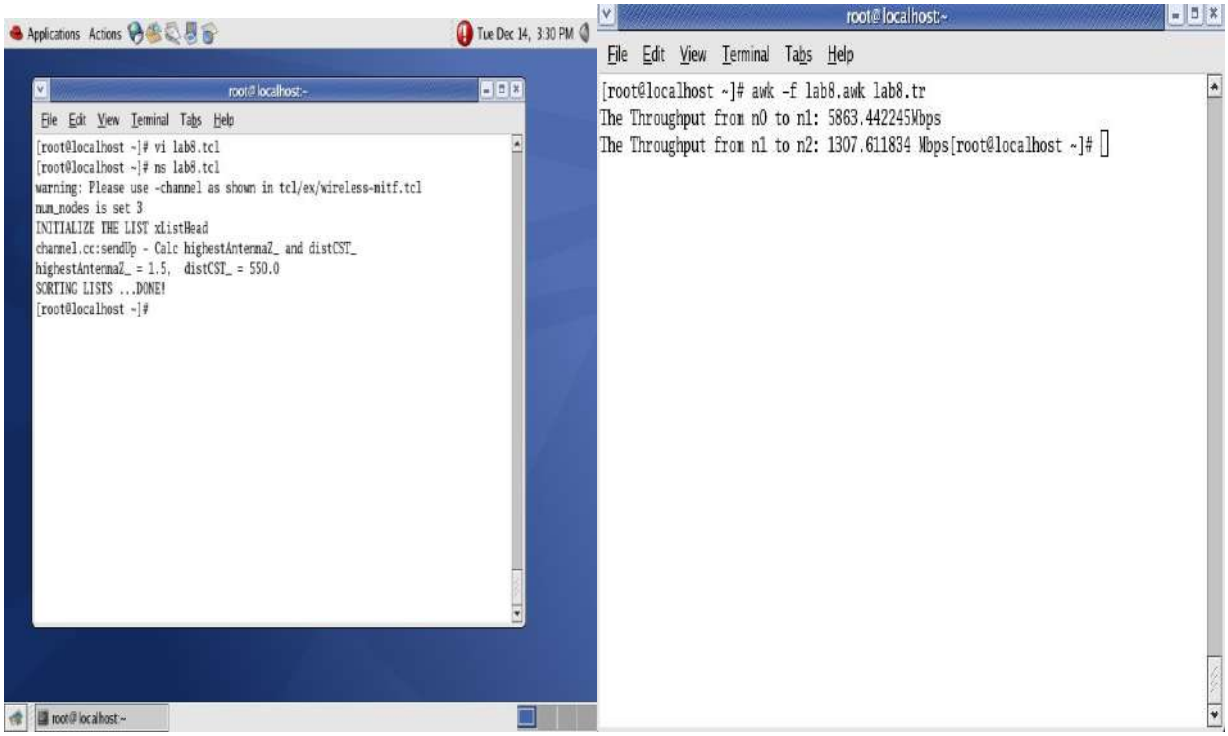


Fig.5.2: ESS with transmission nodes in Wireless LAN

Experiment No: 6

LINK STATE ROUTING ALGORITHM

Objective:

- Implementation of Link state routing algorithm

Theory: The Link State Routing Algorithm is a dynamic routing method used in computer networks to determine the most efficient path for data packets between nodes. Unlike distance-vector algorithms that rely on information from neighboring routers, link state routing allows each router to have a complete and up-to-date map of the entire network topology. This is achieved by having each router discover its neighbors and measure the cost (or metric) of reaching them, then broadcasting this information in the form of Link State Advertisements **(LSAs)** to all other routers in the network using flooding. Once all routers have collected the LSAs, they independently build a complete network graph and use Dijkstra's Shortest Path First (SPF) algorithm to calculate the best path to every other node.

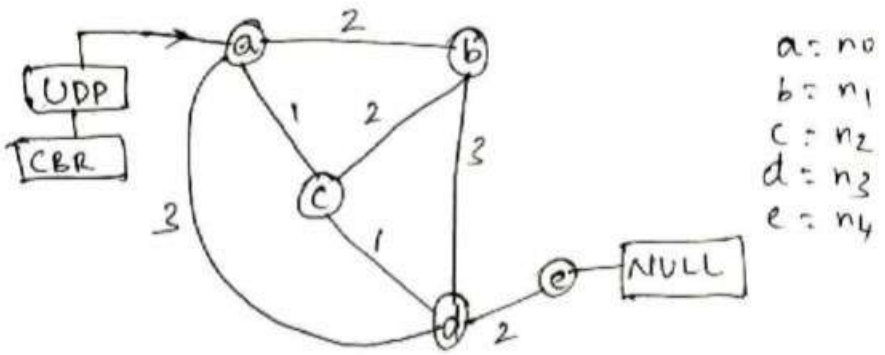


Fig.6.1: Link sate routing

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

#Create a ns simulator
set ns [new Simulator]

#Open the NS trace file
set tracefile [open prg6.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open prg6.nam w]
$ns namtrace-all $namfile

#Create 5 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

#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

#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

#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

#Define a 'finish' procedure
proc finish {} {
   global ns tracefile namfile
   $ns  flush-trace
   close  $tracefile
   close $namfile
   exec nam prg6.nam &
   exit 0
}
$ns at 12 "$val(stop)"
$ns at 11 "finish"
$ns at 10 "$ns halt"
$ns run
```

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++);
      }
```

Steps for execution

- Open vi editor and type program. Program name should have the extension " .tcl "

  [root@localhost ~]# vi lab6.tcl
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys

simultaneously and type “wq” and press Enter key.

- Open vi editor and type awk program. Program name should have the extension “.awk ”

  [root@localhost ~]# vi lab6.awk
- Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press Enter key.
- Run the simulation program

  [root@localhost~]# ns lab6.tcl
  - Here “ns” indicates network simulator. We get the topology shown in the snapshot.
  - Now press the play button in the simulation window and the simulation will begin

- After simulation is completed run awk file to see the output ,

  [root@localhost~]# awk –f lab6.awk lab6.tr
- To see the trace file contents open the file as ,

  [root@localhost~]# vi lab6.tr

Topology

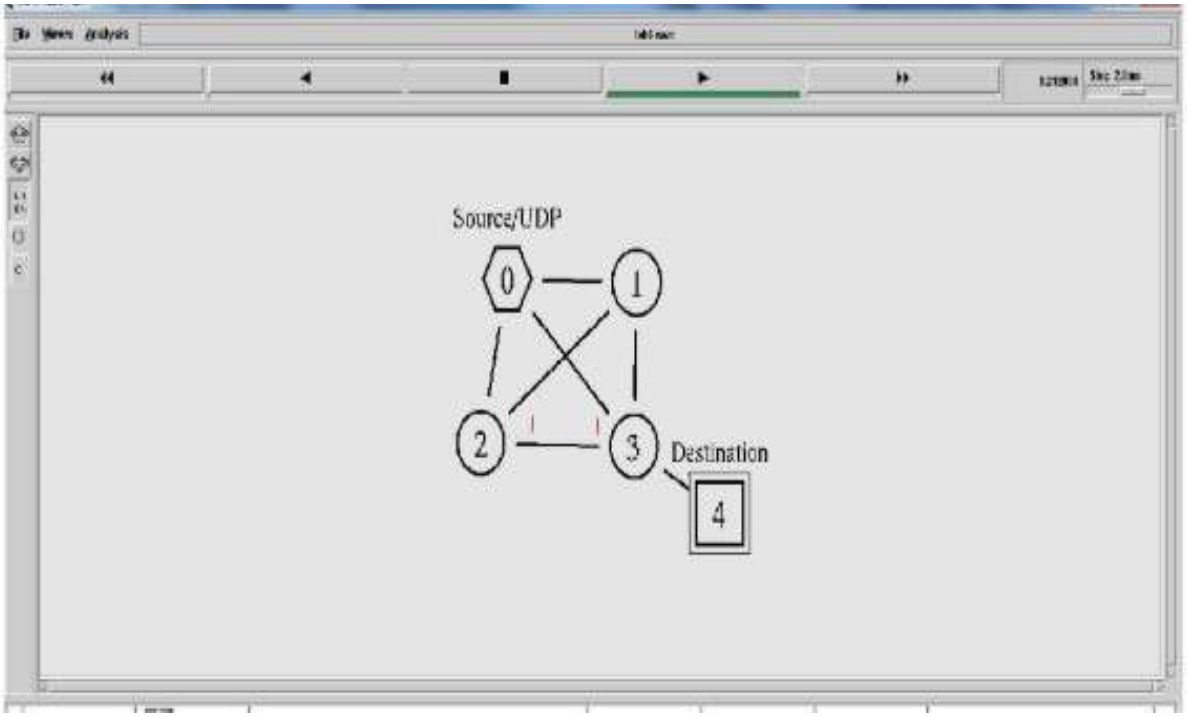


Fig.6.2: Topology link state routing

OUTPUT:

•Total number of routing paths.

1. N0-N1-N2-N3-N4 : Total Cost is **7**
2. N0-N2-N1-N3-N4 : Total Cost is **8**
3. N0-N2-N3-N4 : Total Cost is **4**
4. N0-N3-N4 : Total Cost is **5**

• Shortest according to Link State Algorithm is **N0-N2-N3-N4** having Total Cost is 0

Experiment No: 7

BIT STUFFING and CHARACTER STUFFING

Objectives:

- Write a program for a HLDC frame to perform the following. i) Bit stuffing ii) Character stuffing.

BIT STUFFING

THEORY:

The new technique allows data frames to contain an arbitrary number if bits and allows character codes with an arbitrary no of bits per character. Each frame begins and ends with special bit pattern, 01111110, called a flag byte. Whenever the sender's data link layer encounters five consecutive ones in the data, it automatically stuffs a 0 bit into the outgoing bit stream.

ALGORITHM for BIT STUFFING:

Step1: Input data sequence

Step 2: Add start of frame to output sequence

Step 3: For every bit in input
   a. Append bit to output sequence
   b. Is bit a 1?
      Yes: Increment count,
      If count is 5, append 0 to output sequence and reset count.
      No: Set count to 0

Step 4: Add stop of frame bits to output sequence.

ALGORITHM for BIT DESTUFFING:

Step 1: Input the stuffed sequence.

Step 2: Remove start of frame from sequence.

Step 3: For every bit in input,
   a. Append bit to output sequence.
   b. Is bit a 1?
      Yes: Increment count. If count is 5, remove next bit (which is 0) & reset count.
      No: Set count to 0.

Step 4: Remove end of frame from bits from sequence.

C- LANGUAGE PROGRAM CODE

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

void main() { char ch,array[50]={"01111110"},recd_array[50]; int
counter=0,i=8,j,k;
```

```
    // clrscr();

    printf("Enter the original data stream for bit stuffing:\n");
    while((ch=getchar())!='\n')
    { if(ch=='1')
++counter;
                    else
                    counter=0; array[i+
                    +]=ch; if(counter==5)
                    {
          array[i++]='0';
               counter=0;
                    }
                  }
                  else
                  counter=0;
                        recd_array[k+
                        +]=array[j];
                        if(counter==6)
                        break;
                        else if(counter==5 && array[j+1]=='0')
                        {
                         ++j;
                         counter=0;
                        }
                   }
                  for(j=0;j<=k-strlen("01111110");+
                  +j) printf("%c",recd_array[j]);
                  // getch();
                  }
```

OUTPUT:

Enter the original data stream for bit stuffing:
0011011111000011110

The stuffed data stream is:
01111110001101111100000111100111110

The destuffed data stream is: 0011011111000011110

CHARACTER STUFFING

THEORY:

The framing method gets around the problem of resynchronization after an error by having each frame start with the ASCII character sequence DLE. If the destination ever losses the track of the frame boundaries all it has to do is look for DLE characters to figure out. The data link layer on the receiving end removes the DLE before the data are given to the network layer. This technique is called character stuffing.

ALGORITHM:

Step 1: Start
Step 2 Read the character string to be transmitted in upper case.
Step 3: For stuffing process, append at begin with _DLE‘ as starting flag byte and end with _DLE‘ as ending flag byte of the string.
Step 4: Check the string whether it has _DLE‘.
Step 5: If yes then insert the string _ESC‘ before the character DLE transmit the next character.
Step 6: Continue this process until the completion of string.
Step 7: Stuffed data is obtained.
Step 8: Now destuffing process, remove the appended string at start and end of string.
Step 9: Continue this process until the last character of string.
Step 10: If yes then remove the string _DLE‘ that is encountered first else transmit the data.
Step 11: Continue this process until the last character of string.
Step 12: Original data has been obtained.
Step 13 Stop.

C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
// #include<conio.h>
#include<string.h>
main()
{
 char a[50],s1[100]={"DLE"},s2[10];
 int i,j,l;
 // clrscr();
 printf("\n character stuffing and unstuffing program\n");
 printf("\n @ SENDER --\n");
 printf("\n enter the message to be sent:\n"); gets(a);
 l=strlen(a);  //string length for(i=0;i<l;i+
 +)
 {
  if((a[i]=='D'&&a[i+1]=='L'&&a[i+2]=='E')||
          (a[i]=='E'&&a[i+1]=='S'&&a[i+2]=='C'))
  {
   for(j=l+3;j>=i+3;j--)
```

```
    { a[j]=a[j-3];  //shifted data three position to the right
    }
   l=l+3;
   a[i]='E';
   a[i+1]='S';
   a[i+2]='C';
   i=i+5;
  }
 }
 printf("\n message after character stuffing:\n");
 printf("%s\n",a);
 strcpy(s2,s1);
 strcat(s1,a);
 strcat(s1,s2);
 printf("\n the transmitted frame:\n");
 printf("%s\n",s1);
printf("\n----------------------------------\n");
printf("\n @RECEIVER --\n");
 l=strlen(s1);      //lenght of flag+stuffed_message+flag
 s1[l-3]='\0';      //remove end delimiter(flag)
 l=strlen(s1);
 for(i=0;i<l;i++)
 s1[i]=s1[i+3];     //remove start delimiter(flag)
 l=strlen(s1);
 printf("\nmessage after flag removal at receiver:\n");
 printf("%s\n",s1);
 for(i=0;i<l;i++)
 {
  if(s1[i]=='E'&&s1[i+1]=='S'&&s1[i+2]=='C')
  {
   for(j=i;j<=l;j++)
    { s1[j]=s1[j+3]; //shifted data three position to the left
    }
   l=l-3;
   i=i+2;
  }
 }
 printf("\nmessage after unstuffing:\n");
 printf("%s\n",s1);
 // getch();
}
  }
```

OUTPUT: **Character stuffing and Unstuffing program**

@SENDER –

Enter the message to be sent:
ETXWITHSTXCANDLE

Message after Character stuffing:
ETXWITHSTXCANESCDLE
The transmitted frame:
DLE ETXWITHSTXCANESCDLE **DLE**

……

@RECEIVER –

Message after flag removal at receiver:
ETXWITHSTXCANESCDLE

Message after Unstuffing :

ETXWITHSTXCANDLE

Experiment No: 8

DISTANCE VECTOR ALGORITHM

Objectives:

- Write a program for distance vector algorithm to find suitable path for transmission

THEORY:

Distance Vector Algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbors of its routing table. For each network path, the receiving routers pick the neighbor advertising the lowest cost, then add this entry into its routing table for re-advertisement. To find the shortest path, Distance Vector Algorithm is based on one of two basic algorithms: the Bellman-Ford and the Dijkstra algorithms.

Routers that use this algorithm have to maintain the distance tables (which is a one- dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always upd by exchanging information with the neighboring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.The starting assumption for distance-vector routing is each node knows the cost of the link of each of its directly connected neighbors. Next, every node sends a configured message to its directly connected neighbors containing its own distance table. Now, every node can learn and up its distance table with cost and next hops for all nodes network. Repeat exchanging until no more information between the neighbors.

Consider a node A that is interested in routing to destination H via a directly attached neighbor J. Node A's distance table entry, Dx(Y,Z) is the sum of the cost of the direct-one hop link between A and J, c(A,J), plus neighboring J's currently known minimum-cost path (shortest path) from itself(J) to H. That is Dx(H,J) = c(A,J) + minw{Dj(H,w)} The minw is taken over all the J's. This equation suggests that the form of neighbor-to-neighbor communication that will take place in the DV algorithm - each node must know the cost of each of its neighbors' minimum-cost path to each destination.

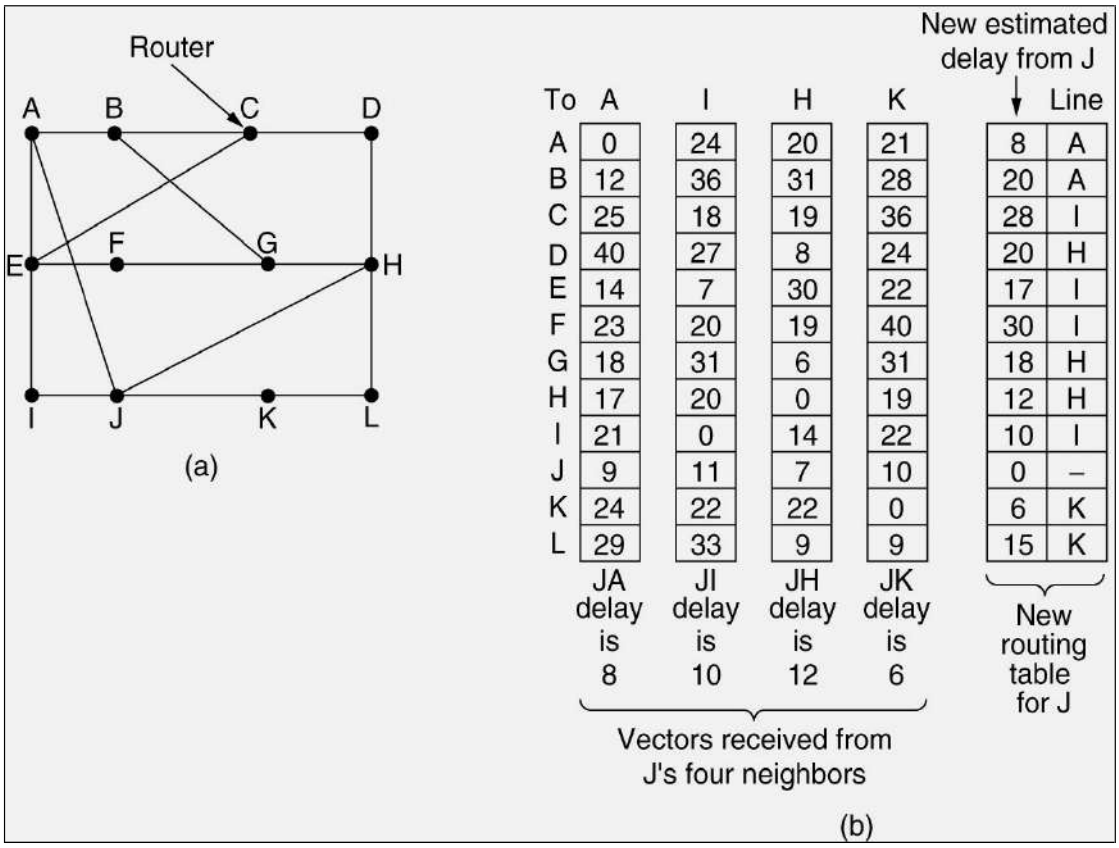


Fig.8: (a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

Implementation Algorithm:

1. send my routing table to all my neighbors whenever my link table changes
2. when I get a routing table from a neighbor on port P with link metric M:
   a. add L to each of the neighbor's metrics
   b. for each entry (D, P', M') in the updated neighbor's table:
      i. if I do not have an entry for D, add (D, P, M') to my routing table
      ii. if I have an entry for D with metric M", add (D, P, M') to my routing table if M' < M"

if my routing table has changed, send all the new entries to all my neighbors

C- LANGUAGE PROGRAM CODE

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

// void rout_table();
int d[10][10],via[10][10];
int i,j,k,l,m,n,g[10][10],temp[10][10],ch,cost;
int main()
{
        system("clear");
        printf("enter the value of no. of nodes\n");
        scanf("%d",&n);
        // rout_table();
```

```
        Printf(—Enter the cost matrix \n‖);
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
             Scanf(‖ %d‖, &g[i][j]);
             if(g[i][j]!=999)
             d[i][j]=1;
            }
        }
          for(i=0;i<n;i++)
          for(j=0;j<n;j++)
           {
            temp[i][j]=g[i][j]; via[i]
            [j]=i;
           }
        while(1)
        {

                for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        if(d[i][j])
                for(k=0;k<n;k++)
                        if(g[i][j]+g[j][k]<g[i][k])
                        {
                                g[i][k]=g[i][j]+g[j][k];
                                via[i][k]=j;
                        }
                for(i=0;i<n;i++)
                {
             printf("table for router %c\n" ,i+97); for(j=0;j<n;j++)
             printf("%c:: %d via %c\n" ,j+97, g[i][j],via[i][j]+97);

                   }
                break;
                }
        }
}
```

OUTPUT:

```
[root@localhost ]# cc prg3.c
[root@localhost ]# ./a.out
enter the value of no. of nodes
4
Enter the routing table:
      |a     b      c      d
---------------------------------------
a     |0     5      1      4
b     |5     0      6      2
c     |1     6      0      3
d     |4     2      3      0
table for router a
a:: 0 via a
b:: 5 via a
c:: 1 via a
d:: 4 via a
table for router b
a:: 5 via b
b:: 0 via b
c:: 5 via d
d:: 2 via b
table for router c
a:: 1 via c
b:: 5 via d
c:: 0 via c
d:: 3 via c

table for router d
a:: 4 via d
b:: 2 via d
c:: 3 via d
d:: 0 via d
do you want to change the cost(1/0)
1
enter the vertices which you want to change the cost
1 3
enter the cost
2
table for router a
a:: 0 via a
b:: 5 via a
c:: 2 via a
d:: 4 via a
table for router b
a:: 5 via b
b:: 0 via b
c:: 5 via d
```

d:: 2 via b

table for router c
a:: 2 via c
b:: 5 via d
c:: 0 via c
d:: 3 via c

table for router d
a:: 4 via d
b:: 2 via b
c:: 3 via d
d:: 0 via d

do you want to change the cost(1/0)
0

Experiment No: 9

SHORTEST PATH USING DIJKSTRA ALGORITHM

Objective:

- To implement Dijkstra's algorithm to compute the shortest routing path.

THEORY:

Dijkstra's algorithm progressively identifies the closest no Dijkstra's algorithm progressively identifies the closest nodes from the source node in order of increasing path cost. The algorithm is iterative. The Dijkstra's algorithm calculates the shortest path between two points on a network using a graph made up of nodes and edges The algorithm divides the nodes into two sets : tentative and permanent . It chooses nodes, makes them tentative, examines them and if they pass the criteria makes them permanent.

ALGORITHM:

Step1: Declare array path [5] [5], min, a [5][5], index, t[5];

Step2: Declare and initialize st=1,ed=5

Step 3: Declare variables i, j, stp, p, edp

Step 4: print —enter the cost —

Step 5: i=1

Step 6: Repeat step (7 to 11) until (i<=5)

Step 7: j=1

Step 8: repeat step (9 to 10) until (j<=5)

Step 9: Read a[i] [j]

Step 10: increment j

Step 11: increment i

Step 12: print —Enter the

path‖ Step 13: read p

Step 14: print —Enter possible paths‖

Step 15: i=1

Step 16: repeat step(17 to 21) until (i<=p)

Step 17: j=1

Step 18: repeat step(19 to 20) until (i<=5)

Step 19: read path[i][j]

Step 20: increment j

Step 21: increment i

Step 22: j=1

Step 23: repeat step(24 to 34) until(i<=p)

Step 24: t[i]=0

Step 25: stp=st

Step 26: j=1

Step 27: repeat step(26 to 34) until(j<=5)

Step 28: edp=path[i][j+1]

Step 29: t[i]= [ti]+a[stp][edp]

Step 30: if (edp==ed) then

Step 31: break;

Step 32: else

Step 33: stp=edp

Step 34: end if

Step 35: min=t[st]

Step 36: index=st

Step 37: repeat step(38 to 41) until (i<=p)

Step 38: min>t[i]

Step 39: min=t[i]

Step 40: index=i

Step 41: end if

Step 42: print‖ minimum cost‖ min

Step 43: print‖ minimum cost pth‖

Step 44: repeat step(45 to 48) until (i<=5)

Step 45: print path[index][i]

Step 46: if(path[idex][i]==ed) then

Step 47: break

Step 48: end if

End

C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
// #include<conio.h>
void main()
{
	int path[5][5], i, j, min, a[5][5], p, st=1,ed=5,stp,edp,t[5],index;
	// clrscr();
	printf("Enter the cost matrix\n");
	for(i=1;i<=5;i++) for(j=1;j<=5;j+
	+) scanf("%d",&a[i][j]);
	printf("Enter the paths\n");
	scanf("%d",&p);
	printf("Enter possible paths\n");
	for(i=1;i<=p;i++) for(j=1;j<=5;j+
	+) scanf("%d",&path[i][j]);
	for(i=1;i<=p;i++)
	{
		t[i]=0;
		stp=st;
		for(j=1;j<=5;j++)
		{
			edp=path[i][j+1]; t[i]=t[i]
			+a[stp][edp]; if(edp==ed)
			break;
			else
			stp=edp;
		}
	}
	min=t[st];index=st;
	for(i=1;i<=p;i++)
	{
		if(min>t[i])
		{
			min=t[i];
			index=i;
		}
	}
	printf("Minimum cost %d",min);
	printf("\n Minimum cost path ");
	for(i=1;i<=5;i++)
	{

		printf("--> %d",path[index][i]);
		if(path[index][i]==ed)
		break;
	}
```

```
        getch();
}
```

OUTPUT:

```
Enter the cost matrix

0   1   4   2   0
1   0   3   7   0
4   3   0   5   0
2   7   5   0   6
0   0   0   6   0
Enter the paths
4
Enter possible paths
1   2   3   4   5
1   2   4   5   0
1   3   4   5   0
1   4   5   0   0
Minimum cost 8
Minimum cost path → 1→ 4 →5
```

Experiment No: 10

ERROR DETECTING CODE Using CRC-CCITT (16-bit)

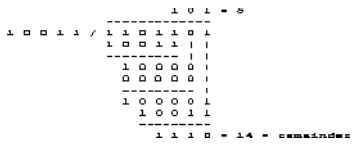
Objectives:

- For the given data, use CRC-CCITT polynomial to obtain CRC code. Verify the program for the cases i)with out error ii)with error.

THEORY :

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The **CRC** calculation or cyclic redundancy check was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it. Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used.

The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation—by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young. It might only look a little bit strange. Also notations differ between countries, but the method is similar.



With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "Lammert"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an exclusive or operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:

- The message bits are appended with *c* zero bits; this *augmented message* is the dividend
- A predetermined *c+1*-bit binary sequence, called the *generator polynomial*, is the divisor
- The checksum is the *c*-bit remainder that results from the division operation

Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

	CRC-CCITT	**CRC-16**	**CRC-32**
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	100000100110000010001110110110111

Table 1. International Standard CRC Polynomials

Error detection with CRC

Consider a message represented by the polynomial M(x)

Consider a generating polynomial G(x)

This is used to generate a CRC = C(x) to be appended to M(x).

Note this G(x) is prime.

Steps: 1. Multiply M(x) by highest power in G(x). i.e. Add So much zeros to M(x).

Divide the result by G(x). The remainder = C(x).

Special case: This won't work if bitstring =all zeros. We don't allow such an M(x).But M(x) bitstring = 1 will work, for example. Can divide 1101 into 1000.

2. If: x div y gives remainder c

   that means: x = n y + c , Hence (x-c) = n y

   (x-c) div y gives remainder 0

   Here (x-c) = (x+c)

   Hence (x+c) div y gives remainder 0
3. Transmit: T(x) = M(x) + C(x)
4. Receiver end: Receive T(x). Divide by G(x), should have remainder 0.

Note if G(x) has order n - highest power is x^n,

then G(x) will cover (n+1) bits

and the remainder will cover n bits. i.e. Add n bits (Zeros) to message.

Some CRC polynomials that are actually used

Some CRC polynomials

- CRC-8:
  x^8+x^2+x+1
  - Used in: 802.16 (along with error *correction*).
- CRC-CCITT:
  $x^{16}+x^{12}+x^5+1$
  - Used in: HDLC, SDLC, PPP default
- IBM-CRC-16
  (ANSI): $x^{16}+x^{15}+x^2+1$
- 802.3:
  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$
  - Used in: Ethernet, PPP rootion

C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
int a[100],b[100],i,j,len,k,count=0;
//Generator Polynomial:g(x)=x^16+x^12+x^5+1
int gp[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,};
int main()
{
        void div();
        system("clear");
        printf("\nEnter the length of Data Frame :");
        scanf("%d",&len);
        printf("\nEnter the Message :");
        for(i=0;i<len;i++)
        scanf("%d",&a[i]);

        //Append r(16) degree Zeros to Msg bits
        for(i=0;i<16;i++)
                a[len++]=0;
        //Xr.M(x)  (ie. Msg+16 Zeros)
        for(i=0;i<len;i++)
                b[i]=a[i];
        //No of times to be divided ie.Msg Length
        k=len-16;
        div();
        for(i=0;i<len;i++)
                b[i]=b[i]^a[i]; //MOD 2 Substraction
        printf("\nData to be transmitted : ");
        for(i=0;i<len;i++)
                printf("%2d",b[i]);
        printf("\n\nEnter the Reveived Data : ");
        for(i=0;i<len;i++)
                scanf("%d",&a[i]);
        div();
        for(i=0;i<len;i++)
                if(a[i]!=0)
                {
                        printf("\nERROR in Recived Data");
                        return 0;
                }
        printf("\nData Recived is ERROR FREE");
}
void div()
{
        for(i=0;i<k;i++)
        {
                if(a[i]==gp[0])
                {
                        for(j=i;j<17+i;j++)
```

```
                    a[j]=a[j]^gp[count++];
                }
                count=0;
        }
}
```

OUTPUT:

Enter the length of Data Frame :4
Enter the Message :1 0 1 1
Data to be transmitted : 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
Enter the Reveived Data : 1 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1
ERROR in Recived Data
Remender is : 0000000100000000

Experiment No: 11(a)

STOP AND WAIT PROTOCOL

Objective:

- To implementation of Stop and Wait Protocol and Sliding Window Protocol

STOP PROTOCOL

THEORY:

If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. The protocol we discuss now is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame.

We still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction, so that we add flow control.

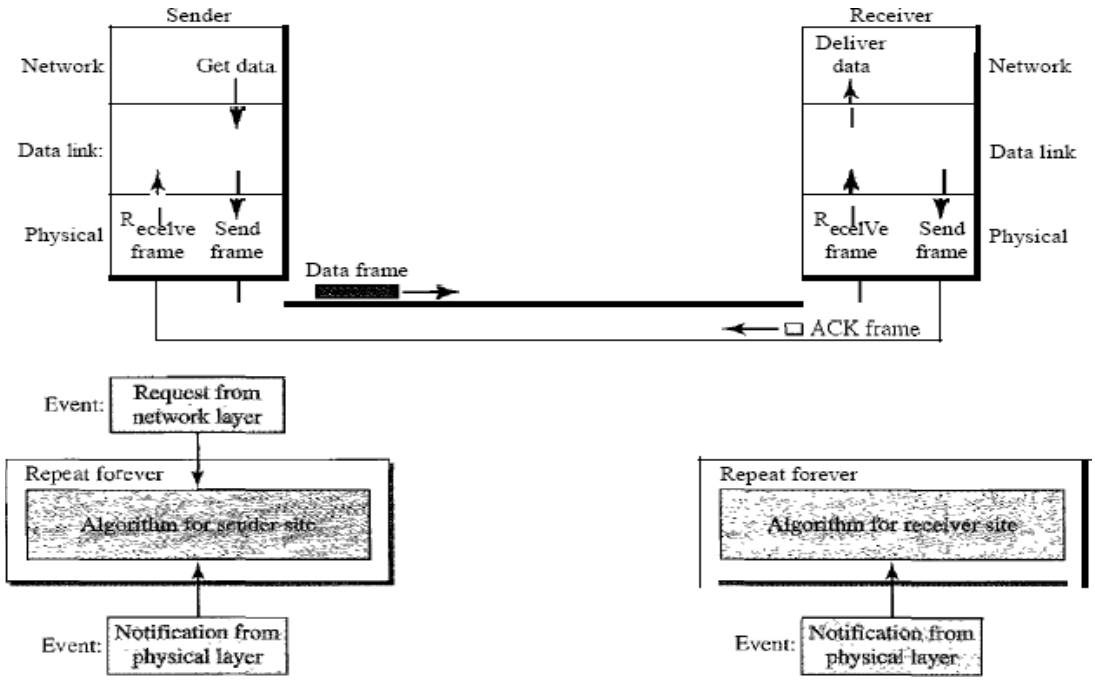


Fig.11(a): Stop Protocol

C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
```

```
{
        int i,j,noframes,x,x1=10,x2;
        noframes=10;
        i=1;  j=1;
                printf("Number of frames is %d ",noframes);
        getch();
        while(noframes>0)
        {
                printf("\n Sending frames is %d",i);
                x=rand()%10;

                if(x%10==0)
                {
                        for(x2=1;x2<2;x2++)
                        {
                                printf("\n Waiting for %d seconds\n",x2);
                                sleep(x2);
                        }
                        printf("\n Sending frames %d\n",i);
                        x=rand()%10;
                }
                printf("\n Ack for frame %d\n",j);
                noframes=noframes-1;
 i++;
 j++;
        }
        printf("\n End of Stop and Wait protocol\n");
}
```

OUTPUT:

```
Number of frames is 10 Sending
        frame is 1
        Ack for frame 1

        Sending frame is 2
        Ack for frame 2

        Sending frame is 3
        Ack for frame 3

        Sending frame is 4
        Waiting for 1 Seconds

        Sending frames 4

        Ack for frame 4

        Sending frame is 5
        Ack for frame 5

        Sending frame is 6
```

Ack for frame 6

Sending frame is 7
Ack for frame 7

Sending frame is 8
Ack for frame 8

Sending frame is 9
Ack for frame 9

Sending frame is 10
Ack for frame 10

End of Stop and Wait protocol

STOP AND WAIT PROTOCOL

THEORY:

It allows multiple frames to be in transmit as compared to stop and wait protocol. In this the receiver has buffer of length W. Transmitter can send up to W frames without ACK. Each frame is numbered according to modular arithmetic. ACK includes number of next frame expected. Sequence number bounded by size of field (k).

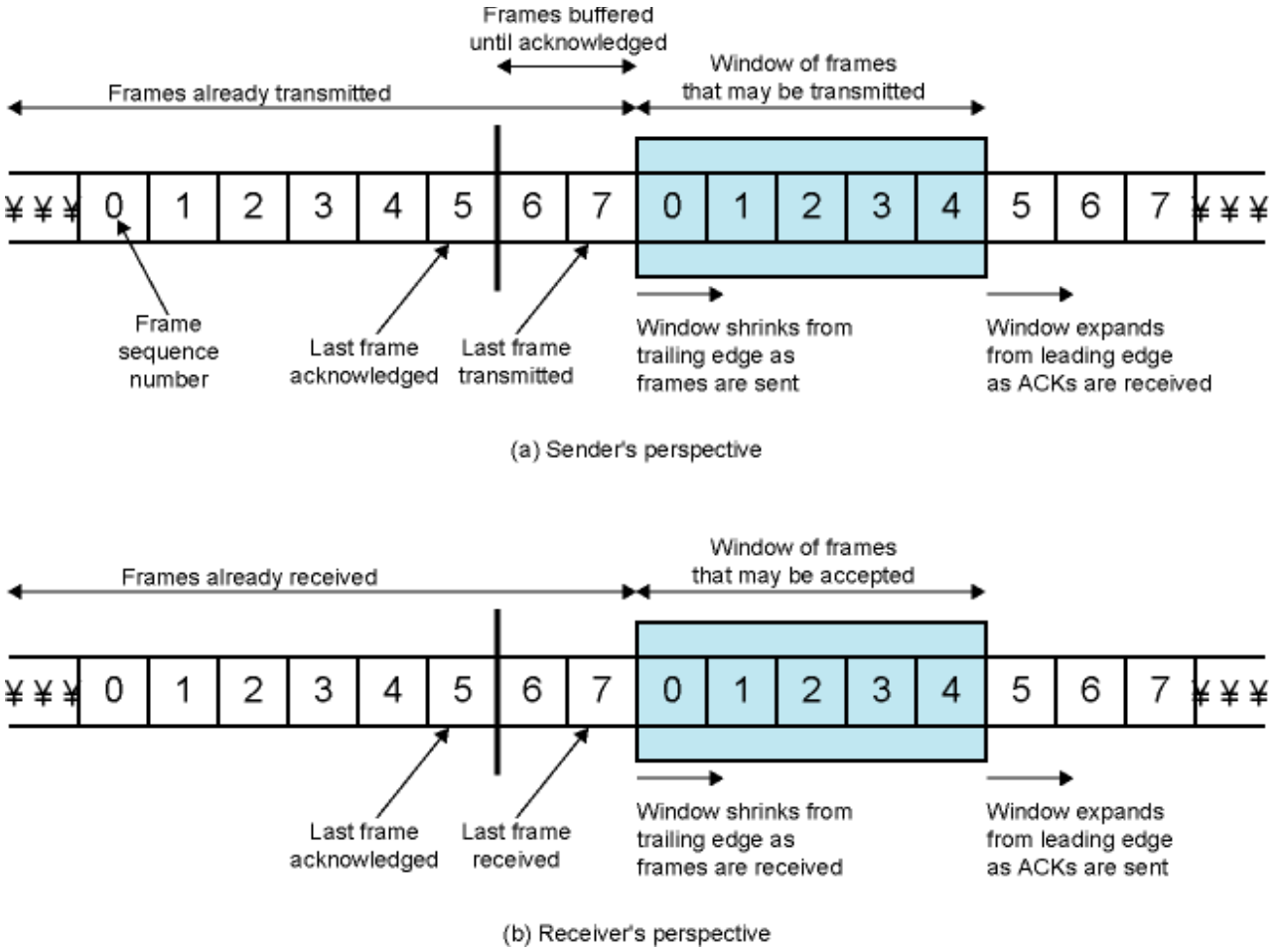


Fig. 11(b): Sender's perspective and Receiver's perspective

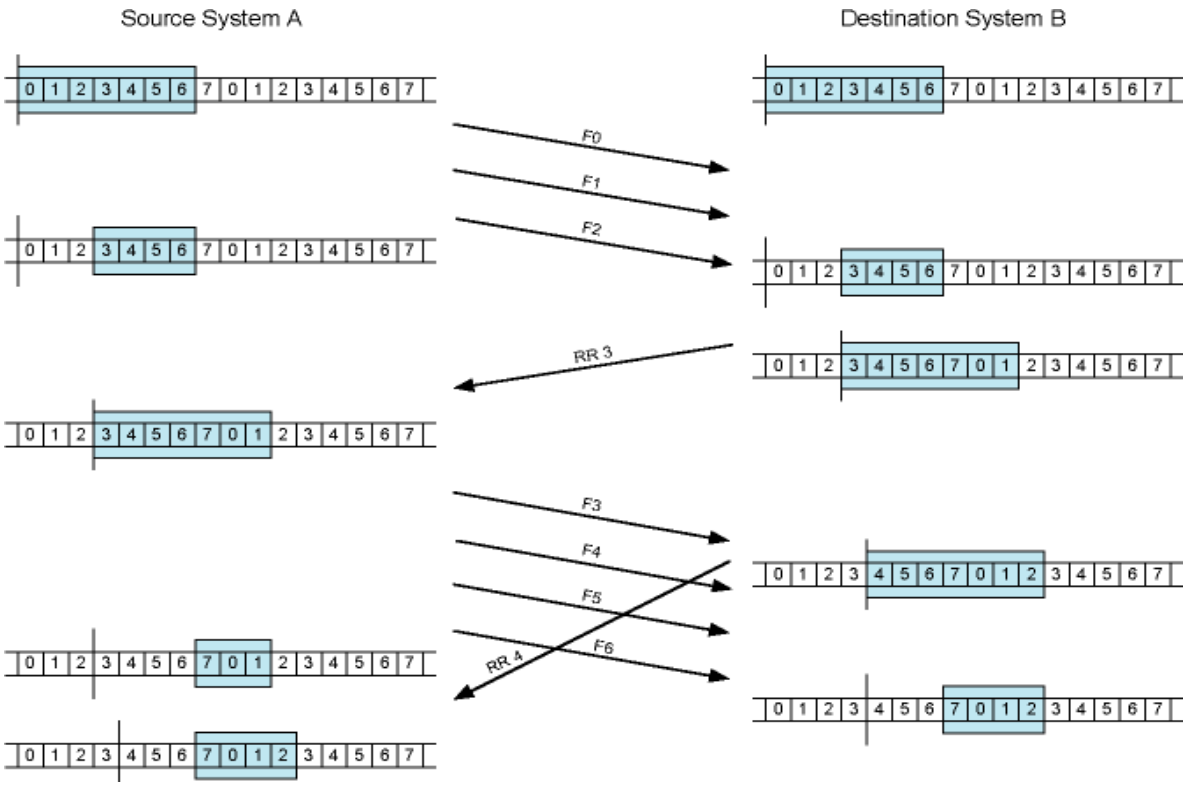


Fig.11(c): Source system A and Source system B of stop and wait protocol

C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
#include<conio.h>
void main()
{
char sender[50],receiver[50];
int i,winsize;
// clrscr();
printf("\n ENTER THE WINDOWS SIZE : ");
scanf("%d",&winsize);
 printf("\n SENDER WINDOW IS EXPANDED TO STORE MESSAGE \n");
 printf("\n ENTER THE DATA TO BE SENT: ");
fflush(stdin);
gets(sender);
for(i=0;i<winsize;i++)
receiver[i]=sender[i];
receiver[i]=NULL;
 printf("\n MESSAGE SEND BY THE SENDER:\n");
 puts(sender);
 printf("\n WINDOW SIZE OF RECEIVER IS EXPANDED\n");
printf("\n ACKNOWLEDGEMENT FROM RECEIVER \n");
for(i=0;i<winsize;i++);
printf("\n ACK:%d",i);
printf("\n MESSAGE RECEIVED BY RECEIVER IS : ");
puts(receiver);
printf("\n WINDOW SIZE OF RECEIVER IS SHRINKED \n");
```

```
getch();
}
```

OUTPUT:

```
ENTER THE WINDOW SIZE: 5
SENDER WINDOW IS EXPANDED TO STORE MESSAGE ENTER
THE DATA TO BE SENT: CITGUBBI
MESSAGE SEND BY THE SENDER: CITGUBBI
WINDOW SIZE OF RECEIVER IS EXPANDED
ACKNOWLEDGEMENT FROM RECEIVER ACK:
5
MESSAGE RECEIVED BY RECEIVER IS: CITGU WINDOW
SIZE OF RECEIVER IS SHRINKED
```

Experiment No: 12

Congestion Control Using LEAKY BUCKET ALGORITHM

Objective:

- To write a program for congestion control using leaky bucket algorithm

THEORY:

The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spillover. In a similar way if the bucket is empty the output will be zero. From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).

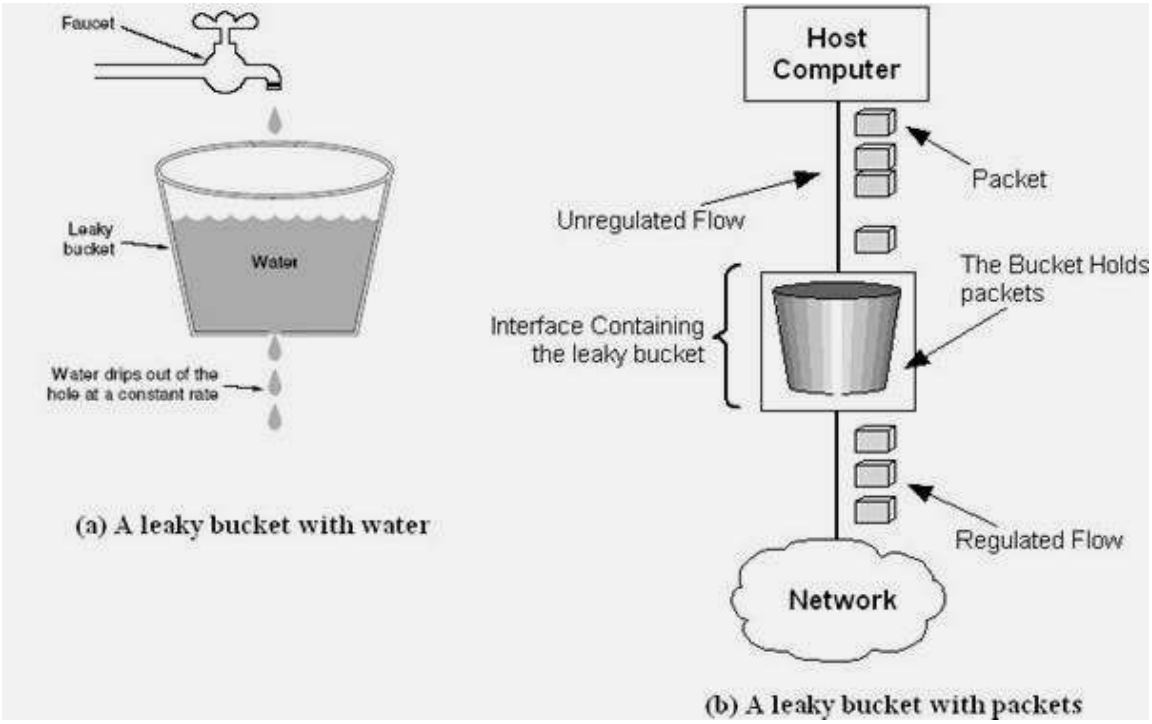


Fig.12: The leaky bucket traffic shaping algorithm

While leaky bucket eliminates completely bursty traffic by regulating the incoming data

flow its main drawback is that it drops packets if the bucket is full. Also, it doesn‘t take into account the idle process of the sender which means that if the host doesn‘t transmit data for some time the bucket becomes empty without permitting the transmission of any packet.

Implementation Algorithm:

1. Read The Data For Packets
2. Read The Queue Size
3. Divide the Data into Packets
4. Assign the random Propagation delays for each packets to input into the bucket (input_packet).
5. wlile((Clock++<5*total_packets)and
   (out_packets< total_paclets))
   a. if (clock == input_packet)
      i. insert into Queue
   b. if (clock % 5 == 0)
      i. Remove paclet from Queue
6. End

C- LANGUAGE PROGRAM CODE

```
#include<stdio.h>
#include<strings.h>
#include<stdio.h>
int min(int x,int y)
{
if(x<y)
    return x;
else
   return y;
}
int main()
{
int drop=0,mini,nsec,cap,count=0,i,inp[25],process;
syste("clear");
printf("Enter The Bucket Size\n");
scanf("%d",&cap);
printf("Enter The Operation Rate\n");
scanf("%d",&process);
printf("Enter The No. Of Seconds You Want To Stimulate\n");
```

```
scanf("%d",&nsec);

for(i=0;i<nsec;i++)
{
printf("Enter The Size Of The Packet Entering At %d

sec\n",i+1);
scanf("%d",&inp[i]);
}
printf("\nSecond|Packet Recieved|Packet Sent|Packet
Left|Packet Dropped|\n");
printf("-------------------------------------------------------------------\n");
for(i=0;i<nsec;i++)
{
count+=inp[i];
if(count>cap)
{
drop=count-cap;
count=cap;
}
printf("%d",i+1);
printf("\t%d",inp[i]);
mini=min(count,process);
printf("\t\t%d",mini);
count=count-mini;
printf("\t\t%d",count);
printf("\t\t%d\n",drop);
drop=0;
}
for(;count!=0;i++)
{
if(count>cap)
{
drop=count-cap;
count=cap;
}
printf("%d",i+1);
printf("\t0");
mini=min(count,process);
printf("\t\t%d",mini);
count=count-mini;
printf("\t\t%d",count);
printf("\t\t%d\n",drop);
}
}
```

OUTPUT:

Compile and run

```
$ cc –o Congestion Congestion.c
$ ./Congestion
Enter The Bucket Size
5
Enter The Operation Rate
2
Enter The No. Of Seconds You Want To Stimulate
3
Enter The Size Of The Packet Entering At 1 sec
5
Enter The Size Of The Packet Entering At 1 sec
4
Enter The Size Of The Packet Entering At 1 sec
3
Second|Packet Recieved|Packet Sent|Packet Left|Packet Dropped|
--------------------------------------------------------------------------------
    1          5          2          3          0
    2          4          2          3          2
    3          3          2          3          1
    4          0          2          1          0
    5          0          1          0          0
```

Virtual Lab
To write a c program using Implementation of Go back-N and Selective Repeat Protocols

Objective:

- To write a c program using Implementation of Go back-N and Selective Repeat Protocols

Algorithm:

STEP 1. Start.
STEP 2. Establish connection (recommended UDP)
STEP 3. Accept the window size from the client(should be <=40)
STEP 4. Accept the packets from the network layer.
STEP 5. Calculate the total frames/windows required.
STEP 6. Send the details to the client(totalpackets,totalframes.)
STEP 7. Initialise the transmit buffer.
STEP 8. Built the frame/window depending on the windowsize.
STEP 9. Transmit the frame.
STEP 10. Wait for the acknowledgement frame.
STEP 11. Close the connection.

Program:

```
#include<stdio.h>
int main()
{
int windowsize,sent=0,ack,i;
printf("enter window size\n");
scanf("%d",&windowsize);
while(1)
{
        for( i = 0; i < windowsize; i++)
                {
                        printf("Frame %d has been transmitted.\n",sent);
                        sent++;
                        if(sent == windowsize)
                        break;
                }
```

```
                printf("\nPlease enter the last Acknowledgement received.\n");
                scanf("%d",&ack);

                if(ack == windowsize)
                        break;
                else
                        sent = ack;
                }
    return 0;
    }
```

OUTPUT:

enter window size 8
Frame 0 has been transmitted
Frame 1 has been transmitted.
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.
Please enter the last Acknowledgement received.
2
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.
Please enter the last Acknowledgement received. 8