JYOTHY INSTITUTE OF TECHNOLOGY

Tataguni, Bangalore-82

Department of Electronics & Communication Engineering

Laboratory Manual

Computer Networks Laboratory

[18ECL76]

[As per Choice Based Credit System (CBCS) scheme]
(Effective from the academic year 2017-18)

SEMESTER –VI

Prepared by:

Mrs. Smita Agarwal
Assistant Professor, Dept. of ECE

Reviewed by:

Mrs. Krishnaveni D
Assistant Professor, Dept. of ECE

VISION & MISSION OF THE DEPARTMENT

Vision:
To be a Department of excellence at a global level in Electronics and Communication Engineering education, incorporating Research & Innovation and Leadership training components.

Mission:
The Department Will,
- ☐ Strive to provide state of Art infrastructure in classrooms and laboratories.
- ☐ Enable all-round development with individual attention and innovative teaching learning methodology.
- ☐ Impart leadership qualities into the students by exposing them to industry and research in global Electronics and Communication Engineering domain.

VISION & MISSION OF THE INSTITUTION

Vision:
To be an Institution of Excellence in Engineering education, Innovation and Research and work towards evolving great leaders for the country's future and meeting global needs.

Mission:
The Institution aims at providing a vibrant, intellectually and emotionally rich teaching learning environment with the State of the Art Infrastructure and recognizing and nurturing the potential of each individual to evolve into ones own self and contribute to the welfare of all.

PROGRAMME EDUCATION OBJECTIVES [PEOs]

PEO1: (Domain Knowledge) Graduates of Electronics & Communication Engineering will be able to utilize mathematics, science, engineering fundamentals, theoretical as well as laboratory based experiences to identify, formulate & solve engineering problems and succeed in advanced engineering or other fields.

PEO2: (Professional Employment) Graduates of Electronics & Communication Engineering will succeed in entry-level engineering positions in VLSI, Communication and Fabrication industries in regional, national, or global industries.

PEO3: (Engineering Citizenship) Graduates of Electronics & Communication Engineering will be prepared to communicate and work effectively on individual & team based engineering projects while practicing the ethics of their profession consistent with a sense of social responsibility.

PEO4: (Lifelong Learning) Graduates of Electronics & Communication Engineering will be equipped to recognize the importance of, and have the skills for, continuous learning to become experts in their domain and enhance their professional attributes

PROGRAMME OUTCOMES [POs]

PO1: Engineering Knowledge: Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
PO2: Problem Analysis: Identify, formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
PO3: Design/ Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.
PO4: Conduct investigations of complex problems using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.
PO5: Modern Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
PO6: The Engineer and Society: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice.
PO7: Environment and Sustainability: Understand the impact of professional engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.
PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.
PO9: Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams and in multi-disciplinary settings.
PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with 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.
PO11: Project Management and Finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12: Life-long Learning: Recognize the need for and have the preparation and ability to Engage in independent and life-long learning in the broadest context of technological changes

PROGRAMME SPECIFIC OBJECTIVES [PSOs]

PSO1: (Knowledge/ Skills) Explore emerging technologies in the field of Electronics & Communication Engineering using the knowledge and skills gained.
PSO2: (Application/Analysis/Problem solving) Apply techniques in different domains to create innovative products and services in the Communication, VLSI, DSP, and Networking.
PSO3: (Value/ Attribute) Work on various platforms as an individual/ team member to develop useful and safe Circuits, PCB, Power Management Systems and Automation for the society and nation.

COURSE OUTCOMES (COs)

17ECL68.1: Use the network simulator for learning and practice of networking algorithms.
17ECL68.2: Illustrate the operations of network protocols and algorithms using C programming.
17ECL68.3: Simulate the network with different configurations to measure the performance parameters.
17ECL68.4: Implement the data link and routing protocols using C programming.

*Levels of Corelation: 1 – Low, 2 – Medium, 3 – High

C0-PO-PSO MAPPING

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
17ECL76.1	3	2			2			2					3	3	2
17ECL76.2	3	3	3	3	3			2					3	3	2
17ECL76.3	3	3	3	3	3			2					3	3	2
17ECL76.4	3	3	2	2	3			2					3	2	2
17ECL76.5	3	3	3	3	3			2					3	3	2

LABORATORY IN-CHARGES

Mr. Rajesh Sudi, Assistant Professor, ECE

DEPARTEMNT ASSESMENT & EVALUATION CELL (DAEC)

Dr. Chandrasekhar K ,Professor, ECE
Dr. Ajjaiaj H B M, Assoc. Professor, ECE

LABORATORY TECHNICIAN

Ms. Ranjith Bhat, ECE

DO'S & DON'TS

Do's

- Conduct yourself in a responsible manner at all times in the laboratory. Don't talk aloud or crack jokes in lab.
- A lab coat should be worn during laboratory experiments.
- Disconnect all the equipment and switch off the power supplies before leaving.
- Observe good housekeeping practices. Replace the materials in proper place after work to keep the lab area tidy.
- Arrange the chairs before leaving the lab.

Don'ts

- Do not come late to the lab.
- Do not enter the lab without permission from the concerned staff.
- Do not sit in groups in front of a single system.
- Do not wander around the room, distract other students, startle other students or interfere with the laboratory experiments of others.
- Do not eat food, drink beverages or chew gum in the laboratory and do not use laboratory glassware as containers for food or beverages.
- Do not open any irrelevant internet sites on lab computer
- Do not use a flash drive on lab computers.
- Do not upload, delete or alter any software on the lab PC.

SYLLABUS

COMPUTER NETWORKS LABORATORY B.E., VI Semester, Electronics & Communication Engineering [As per Choice Based Credit System (CBCS) scheme]			
Subject Code	18ECL76	IA Marks	40
Number of Lecture Hours/Week	01Hr Tutorial (Instructions) + 02 Hours Laboratory = 03	Exam Marks	60
		Exam Hours	03
CREDITS – 02			

Course objectives: This course will enable students to:

1. Choose suitable tools to model a network and understand the protocols at various OSI levels.
2. Design suitable network and simulate using a network simulator tool.
3. Simulate the networking conceots and protocols using C/C++ programming.
4. Model the networks for different configurations and analyse the results.

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 pint 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 HLDC 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.

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

1. Choose suitable tools to model a network.
2. Use network simulator for learning and practice of networking algorithms.
3. Illustrate the operations of network protocols and algorithms using C programming.
4. Simulate the network with different configurations to measure the performance parameters.
5. Implement the data link and routing protocols using C programming

Graduate Attributes (as per NBA)

- Engineering Knowledge.
- Problem Analysis.
- Design/Development of solutions.

Conduct of Practical Examination:

- All laboratory experiments are to be included for practical examination.
- For examination one question from software and one question from hardware or only one hardware experiments based on the complexity to be set.
- Students are allowed to pick one experiment from the lot.
- Strictly follow the instructions as printed on the cover page of answer script for breakup of marks.
- Change of experiment is allowed only once and 15% Marks allotted to the procedure part to be made zero.

LIST OF EXPERIMENTS

OVERVIEW OF COMPUTER NETWORKS LAB

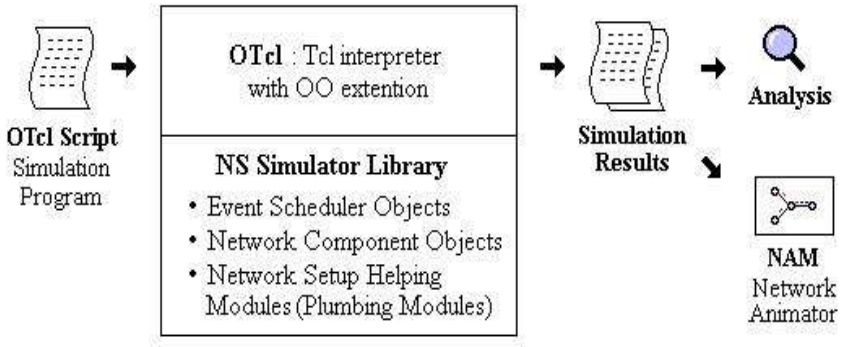
Part A - SIMULATION USING NS-2

Introduction to NS-2:

NS2 is an open-source simulation tool that runs on Linux. It is a discreet event simulator targeted at networking research and provides substantial support for simulation of routing, multicast protocols and IP protocols, such as UDP, TCP, RTP and SRM over wired and wireless (local and satellite) networks.

- 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 – Tool Command Language:

Tcl is a very simple programming language. If you have programmed before, you can learn enough to write interesting Tcl programs within a few hours. This page provides a quick overview of the main features of Tcl. After reading this you'll probably be able to start writing simple Tcl scripts on your own; however, we recommend that you consult one of the many available Tcl books for more complete information.

Basic syntax:

Tcl scripts are made up of commands separated by newlines or semicolons. Commands all have the same basic form illustrated by the following example:

```
expr 20 + 10
```

This command computes the sum of 20 and 10 and returns the result, 30. You can try out this example and all the others in this page by typing them to a Tcl application such as tclsh; after a command completes, tclsh prints its result.

Each Tcl command consists of one or more *words* separated by spaces. In this example there are four words: expr, 20, +, and 10. The first word is the name of a command and the other words are

arguments to that command. All Tcl commands consist of words, but different commands treat their arguments differently. The expr command treats all of its arguments together as an arithmetic expression, computes the result of that expression, and returns the result as a string. In the expr command the division into words isn't significant: you could just as easily have invoked the same command as

```
expr 20+10
```

However, for most commands the word structure is important, with each word used for a distinct purpose. All Tcl commands return results. If a command has no meaningful result, then it returns an empty string as its result.

Variables

Tcl allows you to store values in variables and use the values later in commands. The set command is used to write and read variables. For example, the following command modifies the variable x to hold the value 32:

```
set x 32
```

The command returns the new value of the variable. You can read the value of a variable by invoking set with only a single argument:

```
set x
```

You don't need to declare variables in Tcl: a variable is created automatically the first time it is set. Tcl variables don't have types: any variable can hold any value.

To use the value of a variable in a command, use *variable substitution* as in the following example:

```
expr $x*3
```

When a $ appears in a command, Tcl treats the letters and digits following it as a variable name, and substitutes the value of the variable in place of the name. In this example, the actual argument received by the expr command will be 32*3 (assuming that variable x was set as in the previous example). You can use variable substitution in any word of any command, or even multiple times within a word:

```
set cmd expr
set x 11
$cmd $x*$x
```

Command substitution:

You can also use the result of one command in an argument to another command. This is called *command substitution*:

```
set a 44
set b [expr $a*4]
```

When a [appears in a command, Tcl treats everything between it and the matching] as a nested Tcl command. Tcl evaluates the nested command and substitutes its result into the enclosing command in place of the bracketed text. In the example above the second argument of the second set command will be 176.

Quotes and braces:

Double-quotes allow you to specify words that contain spaces. For example, consider the following script:

```
set x 24
set y 18
set z "$x + $y is [expr $x + $y]"
```

After these three commands are evaluated variable z will have the value 24 + 18 is 42. Everything between the quotes is passed to the set command as a single word. Note that (a) command and variable substitutions are performed on the text between the quotes, and (b) the quotes themselves are not passed to the command. If the quotes were not present, the set command would have received 6 arguments, which would have caused an error.

Curly braces provide another way of grouping information into words. They are different from quotes in that no substitutions are performed on the text between the curly braces:

```
set z {$x + $y is [expr $x + $y]}
```

This command sets variable z to the value "$x + $y is [expr $x + $y]".

Control structures:

Tcl provides a complete set of control structures including commands for conditional execution, looping, and procedures. Tcl control structures are just commands that take Tcl scripts as arguments. The example below creates a Tcl procedure called power, which raises a base to an integer power:

```
proc power {base p} {
set result 1
while {$p > 0} {
set result [expr $result * $base]
set p [expr $p - 1]
}
return $result
}
```

This script consists of a single command, proc. The proc command takes three arguments: the name of a procedure, a list of argument names, and the body of the procedure, which is a Tcl script. Note that everything between the curly brace at the end of the first line and the curly brace on the last line is passed verbatim to proc as a single argument. The proc command creates a new Tcl command named power that takes two arguments. You can then invoke power with commands like the following:

```
power 2 6
power 1.15 5
```

When power is invoked, the procedure body is evaluated. While the body is executing it can access its arguments as variables: base will hold the first argument and p will hold the second.

The body of the power procedure contains three Tcl commands: set, while, and return. The while command does most of the work of the procedure. It takes two arguments, an expression ($p > 0) and a body, which is another Tcl script. The while command evaluates its expression argument

using rules similar to those of the C programming language and if the result is true (nonzero) then it evaluates the body as a Tcl script. It repeats this process over and over until eventually the expression evaluates to false (zero). In this case the body of the while command multiplied the result value by base and then decrements p. When p reaches zero the result contains the desired power of base. The return command causes the procedure to exit with the value of variable result as the procedure's result

Where do commands come from?

As you have seen, all of the interesting features in Tcl are represented by commands. Statements are commands, expressions are evaluated by executing commands, control structures are commands, and procedures are commands.

Tcl commands are created in three ways. One group of commands is provided by the Tcl interpreter itself. These commands are called *builtin commands*. They include all of the commands you have seen so far and many more (see below). The builtin commands are present in all Tcl applications.

The second group of commands is created using the Tcl extension mechanism. Tcl provides APIs that allow you to create a new command by writing a *command procedure* in C or C++ that implements the command. You then register the command procedure with the Tcl interpreter by telling Tcl the name of the command that the procedure implements. In the future, whenever that particular name is used for a Tcl command, Tcl will call your command procedure to execute the command. The builtin commands are also implemented using this same extension mechanism; their command procedures are simply part of the Tcl library.

When Tcl is used inside an application, the application incorporates its key features into Tcl using the extension mechanism. Thus the set of available Tcl commands varies from application to application. There are also numerous extension packages that can be incorporated into any Tcl application. One of the best known extensions is Tk, which provides powerful facilities for building graphical user interfaces. Other extensions provide object-oriented programming, database access, more graphical capabilities, and a variety of other features. One of Tcl's greatest advantages for building integration applications is the ease with which it can be extended to incorporate new features or communicate with other resources.

The third group of commands consists of procedures created with the proc command, such as the power command created above. Typically, extensions are used for lower-level functions where C programming is convenient, and procedures are used for higher-level functions where it is easier to write in Tcl.

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:

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

Features of NS2:

NS2 can be employed in most Unix systems and windows. Most of the NS2 code is in C++. It uses TCL as its scripting language, Otcl adds object orientation to TCL.NS(version 2) is an object oriented, discrete event driven network simulator that is freely distributed and open source.

- Traffic Models: CBR, VBR, Web etc.
- Protocols: TCP, UDP, HTTP, Routing algorithms, MAC etc.
- Error Models: Uniform, bursty etc.
- Misc.: Radio propagation, Mobility models, Energy Models
- Topology Generation tools
- Visualization tools (NAM), Tracing

Structure of NS:

- NS is an object oriented discrete event simulator
    - Simulator maintains list of events and executes one event after another
    - Single thread of control: no locking or race conditions
- Back end is C++ event scheduler
    - Protocols mostly
    - Fast to run, more control
- Front end is OTCL
    - Creating scenarios, extensions to C++ protocols
    - fast to write and change

Platforms:

It can be employed in most Unix systems (FreeBSD, Linux, Solaris) and Windows.

Source code:

Most of NS2 code is in C++

Scripting language:

It uses TCL as its scripting language OTcl adds object orientation to TCL.

Protocols implemented in NS2:

- Transport layer (Traffic Agent) – TCP, UDP
- Network layer (Routing agent)
- Interface queue – FIFO queue, Drop Tail queue, Priority queue

- Logic link control layer – IEEE 802.2, AR

How to use NS2:

- Design Simulation – Determine simulation scenario
- Build ns-2 script using tcl.
- Run simulation

Simulation with NS2:

- Define objects of simulation.
- Connect the objects to each other
- Start the source applications. Packets are then created and are transmitted through network.
- Exit the simulator after a certain fixed time.

NS programming Structure

- Create the event scheduler
- Turn on tracing
- Create network topology
- Create transport connections
- Generate traffic
- Insert errors

Sample Wired Simulation using NS-2:

Creating Event Scheduler

- Create event scheduler: set ns [new simulator]
- Schedule an event: $ns at <time> <event>
    - event is any legitimate ns/tcl function

```
$ns at 5.0 "finish"
proc finish {} {
global ns nf
close $nf
exec nam out.nam &
exit 0
}
```

- Start Scheduler

```
$ns run
```

Tracing:

- All packet trace

```
$ns traceall[open out.tr w]
<event> <time> <from> <to> <pkt> <size>…
<flowid> <src> <dst> <seqno> <aseqno>
+ 0.51 0  1 cbr 500 —— 0 0.0 1.0      0   2
_ 0.51 0  1 cbr 500 —— 0    0.0 1.0   0   2
```

R 0.514 0 1 cbr 500 ——0 0.0 1.0 0 0

- Variable trace

```
set par [open output/param.tr w]
$tcp attach $par
$tcp trace cwnd_
$tcp trace maxseq_
$tcp trace rtt_
```

Tracing and Animation:

- Network Animator

```
set nf [open out.nam w]
$ns namtraceall
$nf
proc finish {} {
global ns nf
close $nf
exec nam out.nam &
exit 0
}
```

Creating topology:

- Two nodes connected by a link
- Creating nodes

```
set n0 [$ns node]
set n1 [$ns node]
```

- Creating link between nodes

$ns <link_type> $n0 $n1 <bandwidth> <delay><queue-type> $ns duplex-link$n0 $n1 1Mb 10ms DropTail

Data Sending:

- Create UDP agent

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
```

- Create CBR traffic source for feeding into UDP agent set cbr0

[new Application/Traffic/CBR]

$cbr0 set packetSize_ 500 $cbr0 set interval_ 0.005 $cbr0 attach-agent$udp0

- Create traffic sink

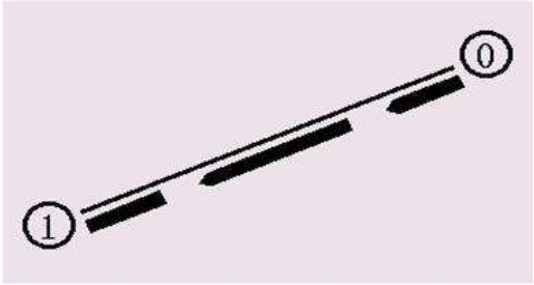
```
set null0 [new Agent/Null]
$ns attach-agent$n1 $null0
```

- Connect two agents

$ns connect $udp0 $null0

- Start and stop of data

$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"



Traffic on top of TCP:

- FTP

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

- Telnet

set telnet [new Application/Telnet]
$telnet attach-agent$tcp0

PROCEDURE:

STEP 1: Start

STEP 2: Create the simulator object ns for designing the given simulation

STEP 3: Open the trace file and nam file in the write mode

STEP 4: Create the nodes of the simulation using the 'set' command

STEP 5: Create links to the appropriate nodes using $ns duplex-link command

STEP 6: Set the orientation for the nodes in the simulation using 'orient' command

STEP 7: Create TCP agent for the nodes and attach these agents to the nodes

STEP 8: The traffic generator used is FTP for both node0 and node1

STEP 9: Configure node1 as the sink and attach it

STEP10: Connect node0 and node1 using 'connect' command

STEP 11: Setting color for the nodes

STEP 12: Schedule the events for FTP agent 10 sec

STEP 13: Schedule the simulation for 5 minutes

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	To	PKT	PKT	Flags	Fid	Src	Dest	Seq	Pkt
		Node	Node	Type	Size			Addr	Addr	Num	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)

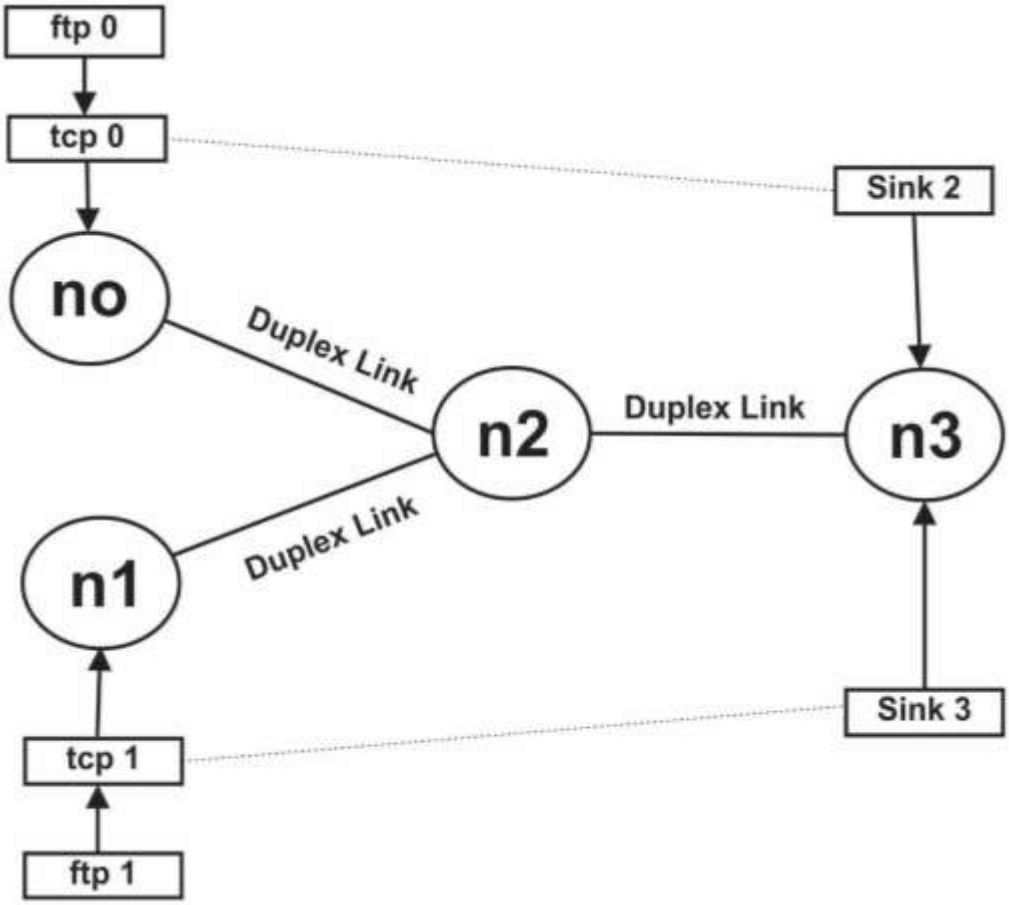
This is the unit name for the y-axis. Its default is —Y‖.

PART-A

EXPERIMENTS-1

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

Design:



Note:

1. Set the queue size fixed from n0 to n2 as 10, n1-n2 to 10 and from n2-n3 as 5. Syntax: To set the queue size
   $ns set queue-limit <from> <to>
   <size> Eg: $ns set queue-limit $n0 $n2
   10
2. Go on varying the bandwidth from 10, 20 30. . and find the number of packets dropped at the node 2 as shown in the Fig 1.
3. To add the agents between the nodes

i). set udp0 [new Agent/UDP]
   $ns attach-agent $n0 $udp0
ii). set cbr0 [new Application/Traffic/CBR]
   $cbr0 attach-agent $udp0
iii). set null [new Agent/Null]

```
$ns attach-agent$n3   $null
Finally, $ns connect $udp0 $null
set ns [ new Simulator]
set tf [ open lab1.tr w ]
$ns trace-all $tf
set nf [ open lab1.nam w ]
$ns namtrace-all $nf
```

 # The below code is used to create the nodes.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

#This is used to give color to the packets.

```
$ns color 1 "red"
$ns color 2 "blue"
$n0 label "Source/udp0"
$n1 label "Source/udp1"
$n2 label "Router"
$n3 label "Destination/Null"
```

#Vary the below Bandwidth and see the number of packets dropped.

```
$ns duplex-link $n0 $n2 10Mb 300ms DropTail
$ns duplex-link $n1 $n2 10Mb 300ms DropTail
$ns duplex-link $n2 $n3 1Mb 300ms DropTail
```

#The below code is used to set the queue size b/w the nodes

```
$ns set queue-limit $n0 $n2 10 $ns set queue-limit $n1 $n2 10 $ns set queue-limit $n2 $n3 5
```

#The below code is used to attach an UDP agent to n0, UDP #agent to n1 and null agent to n3.

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set null3 [new Agent/Null]
$ns attach-agent $n3 $null3
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
```

#The below code sets the udp0 packets to red and udp1 #packets to blue color

```
$udp0 set class_ 1
$udp1 set class_ 2
```

#The below code is used to connect the agents.

```
$ns connect $udp0 $null3
$ns connect $udp1 $null3
```

#The below code is used to set the packet size to 500

```
$cbr1 set packetSize_ 500Mb
```

#The below code is used to set the interval of the packets, #i.e., Data rate of the packets. if the data rate is high #then packets drops are high.

```
$cbr1 set interval_ 0.005
proc finish { } {
```

```
global ns nf tf
$ns flush-trace
exec nam lab1.nam &
close $tf
close $nf
exit 0
}
$ns at 0.1 "$cbr0 start"
$ns at 0.1 "$cbr1 start"
$ns at 10.0 "finish"
$ns run
```

AWK Script:

```
BEGIN
{
#include<stdio.h>
count=0;
}
{
if($1=="d") #d stands for the packets drops.
count++
}
END
{
printf("The Total no of Packets Dropped due to Congestion :
%d\n\n", count)
}
```

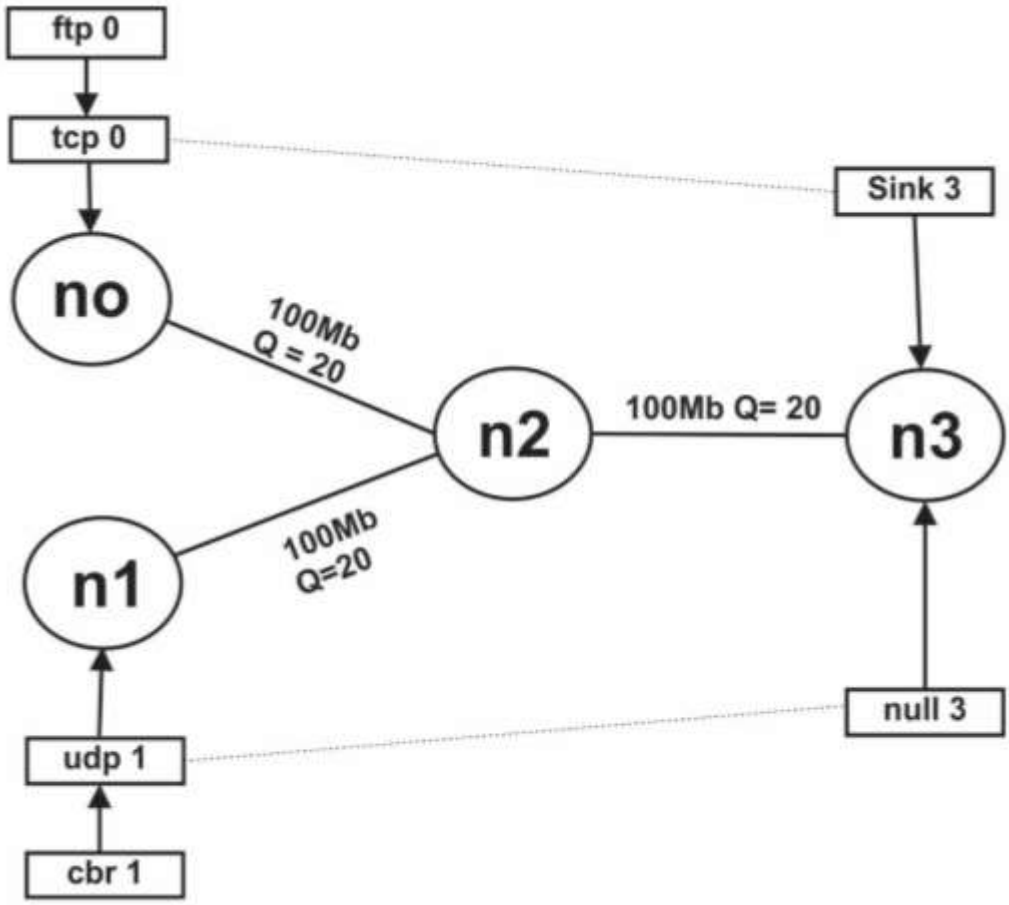
Output:

```
ns lab1.tcl
awk –f lab1.awk lab1.tr
The Total no of packets Dropped due to congestion:4560
```

EXPERIMENTS-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.

Design:



Note:

1.Calculate the total number of packets sent by TCP and UDP by changing the bandwidth from 100Mb,25Mb or 200mb in the Topology and the data rate from 0.1, 0.001 as shown below.

Eg:

Bandwidth	Data Rate	TCP pkt sent	UDP pkt sent
100Mb	0.1	2000	2500
200Mb	0.001	3400	6700

```
set ns [new Simulator]
set tf [open lab2.tr w]
$ns trace-all $tf
set nf [open lab2.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#  The below code is used to set the color and name's to the #nodes.
$ns color 1 "red"
$ns color 2 "blue"
$n0 label "source/TCP"
$n1 label "source/UDP"
$n2 label "Router"
$n3 label "destination"
$ns duplex-link $n0 $n2 100Mb 1ms DropTail $ns duplex-link $n1 $n2
100Mb 1ms DropTail $ns duplex-link $n2 $n3 100Mb 1ms DropTail

#The below code is used to set the color and labels to the #links.

$ns duplex-link-op $n0          $n2 color "green"
$ns duplex-link-op $n0          $n2 label "from 0-2"
$ns duplex-link-op $n1          $n2   color "green"
$ns duplex-link-op $n1          $n2   label "from 1-2"
$ns duplex-link-op $n2          $n3   color "green"
$ns duplex-link-op $n2          $n3   label "from 2-3"
#       The below code is used create TCP and UDP agents and the traffic ftp & cbr respectively.
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink3 [new Agent/TCPSink]
```

```
$ns attach-agent $n3 $sink3
 set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR] $cbr1 attach-agent
$udp1 set null3 [new Agent/Null]

$ns attach-agent $n3 $null3

#The below code is used to set the packet size of ftp and #udp.
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.001
#The below code is used to increase the data rate (if the #interval is more than the more number of packets goes to #destination).
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.001
#This code is used give a color red->tcp and blue ->udp.
$tcp0 set class_ 1
$udp1 set class_ 2
# The below code is used connect the agents. $ns connect $tcp0 $sink3
$ns connect $udp1 $null3

proc finish { } {
                global ns nf tf
                $ns flush-trace

                exec nam lab2.nam &
                close $nf

                close $tf
                exit 0
}
$ns at 0.1 "$cbr1 start"
$ns at 0.2 "$ftp0 start"
$ns at 5.0 "finish"
$ns run
```

AWK Script:

```
BEGIN{
#include<stdio.h>
tcp=0;
udp=0;
}
{
        if($1=="r"&&$3=="2"&&$4=="3"&& $5=="tcp")
                tcp++;
        if($1=="r"&&$3=="2"&&$4=="3"&&$5=="cbr")
                udp++;
}
END{
printf("\n Total number of packets sent by TCP : %d\n",tcp);
printf("\n Total number of packets sent by UDP : %d\n",udp);
}
```

Output:

ns lab2.tcl

awk –f lab2.awk lab2.tr

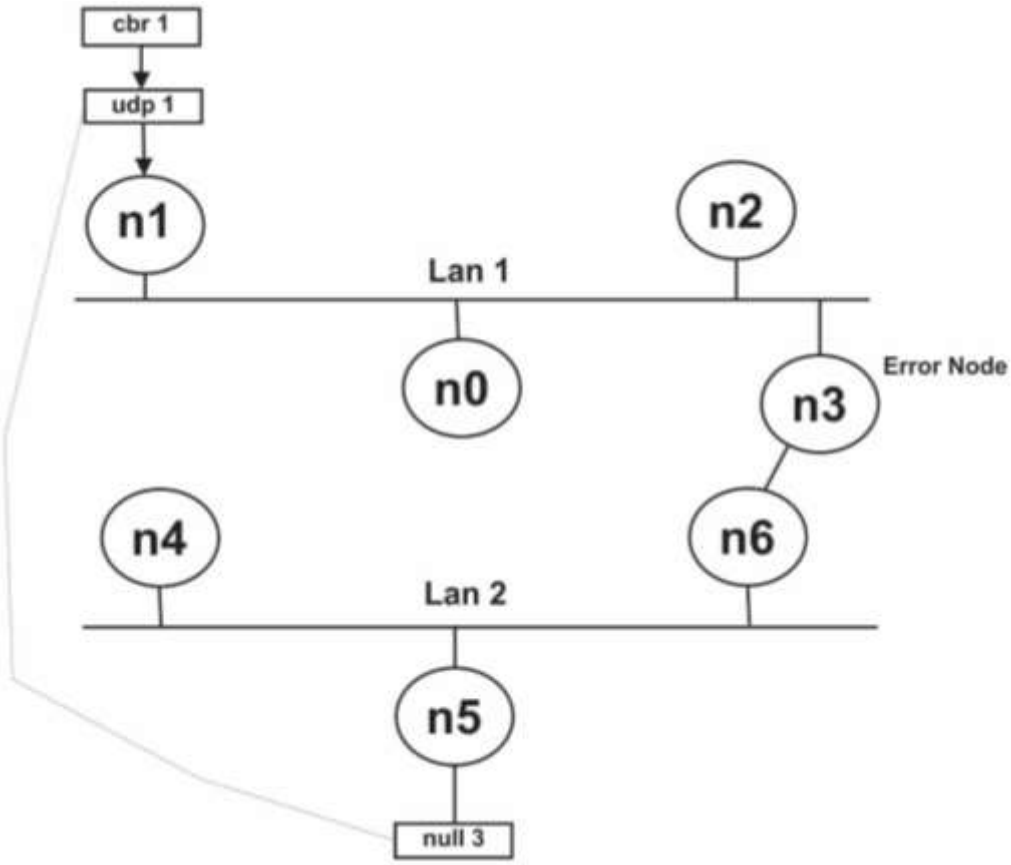
Total number of packets sent by TCP: 1200

Total number of packets sent by UDP: 4500

EXPERIMENTS-3

Implement an Ethernet LAN using n nodes (6-10), change the error rate and data rate and compare the throughput.

Design:



Note:

1. The lan can be created by using the command:

   $ns make-lan " $n0 $n1 $n2 $n3 " 100Mb 10ms LL Queue/DropTail Mac/802_

2. The Error between the nodes n3 and n6 can be added as follows:

   Set err [new ErrorModel]

   $ns lossmodel $err $n3 $n6

   $err set rate_ 0.1 # used to set error rate.

3. The throughput can analyzed by changing the datarate and errorrate as shown below.

First, Fix the data rate to 0.0001 and vary the error rate then throughput decreases as shown below.

Eg:

Error rate	Data rate	Throughput
0.1	0.1	78Mbps
0.2	0.01	89Mbps
0.1	0.001	100Mbps
0.1	0.0001	148Mbps

Now Fix the error rate to 0.1 and vary the data rate then throughput increases as shown below.

Program:

```
set ns [new Simulator]
set tf [open lab5.tr w]
$ns trace-all $tf
set nf [open lab5.nam w]
$ns namtrace-all $nf
$ns color 1 "red"
set n0 [$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]
$n1 label "Source/UDP"
$n3 label "Error Node"
$n5 label "Destination"
```

#The below code is used to create a two Lans (Lan1 and #Lan2).

```
$ns make-lan "$n0 $n1 $n2 $n3" 10Mb 10ms LL Queue/DropTail Mac/802_3
$ns make-lan "$n4 $n5 $n6" 10Mb 10ms LL Queue/DropTail Mac/802_3
```

#The below code is used to connect node n3 of lan1 and n6 of lan2.

```
$ns duplex-link $n3 $n6 100Mb 10ms DropTail
```

```
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

set cbr1 [ new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set null5 [new Agent/Null]
$ns attach-agent $n5 $null5
$ns connect $udp1 $null5
$cbr1 set packetSize_ 1000
$cbr1 set interval_ 0.0001

# This is the data rate. Change; # this to increase the rate.

$udp1 set class_ 1

# The below code is used to add an error model between the #nodes n3 and n6.

set err [new ErrorModel]
$ns lossmodel $err $n3 $n6
$err set rate_ 0.2;

# This is the error rate. Change this ;#rate to add errors between n3 and n6.

    proc finish { } {
global nf ns tf
exec nam lab5.nam &
close $nf
close $tf
exit 0
}
    $ns at 5.0 "finish"
    $ns at 0.1 "$cbr1 start"
    $ns run
```

AWK Script:

```
BEGIN
{
        #include <stdio.h>
        pkt=0;
        time=0
}
{
        if($1="r" && $3=="8" && $4=="5")
        {
                pkt=pkt+$6
                time=$2
        }
}
END
{
printf(" Throughput: %fMbps\n\n",(pkt/time)*(8/1000000));
}
```

Output:

ns lab5.tcl

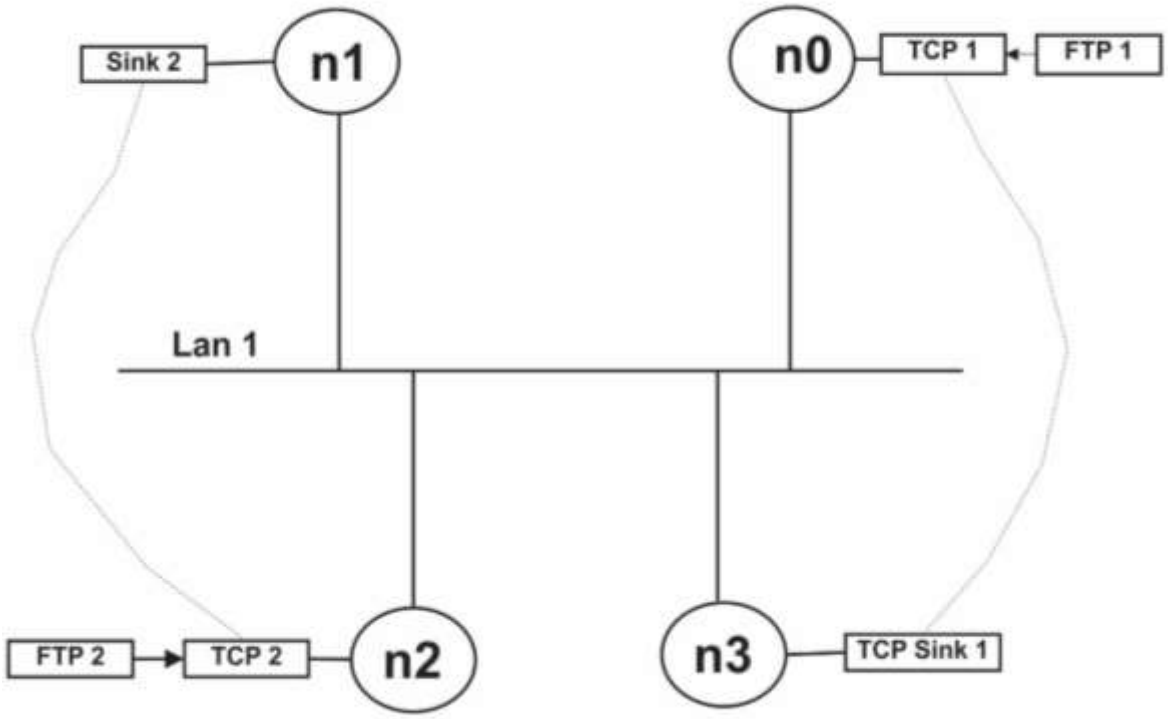
awk –f lab5.awk lab5.tr

Throughput:148Mbps

EXPERIMENTS-4

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

Design:



The congestion at the nodes can be traced by the following steps:

Step 1: Create a file1.tr and file2.tr

set file1 [open file1.tr w] set file2 [open file2.tr w]

Step 2: Attach these files (file1.tr & file2.tr) to the agents tcp0 and tcp1 as shown below.

$tcp0 attach $file1 $tcp1 attach $file2

Step 3: To trace the congestion window value we use the trace command to trace the congestion window values.

$tcp0 trace cwnd_ $tcp1 trace cwnd_

Then, the congestion window values are stored in the file1 and file2.

Step 4: To plot the graph the steps are given below

ns lab7.tcl

awk –f lab7.awk file1.tr > tcp1

awk –f lab7.awk file2.tr > tcp2

xgraph –x "Time" -y "cwnd" tcp1 tcp2

Program:

```
set ns [new Simulator]
set tf [open lab7.tr w]
$ns trace-all $tf
set nf [open lab7.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]

$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 sink2 [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
```

AWK Script:

```
BEGIN{
        #include<stdio.h>
}
{
  if($6=="cwnd_")
      printf("%f \t %f \n", $1,$7);
}
END
{
        puts "DONE"
}
```

Output:

ns lab7.tcl

awk –f lab7.awk file1.tr>tcp1

awk –f lab7.awk file2.tr>tcp2

xgraph –x "time" –y "convalue " tcp1 tcp2

Note: To set the foreground and background color of the graph choose the appropriate options by giving **xgraph –**

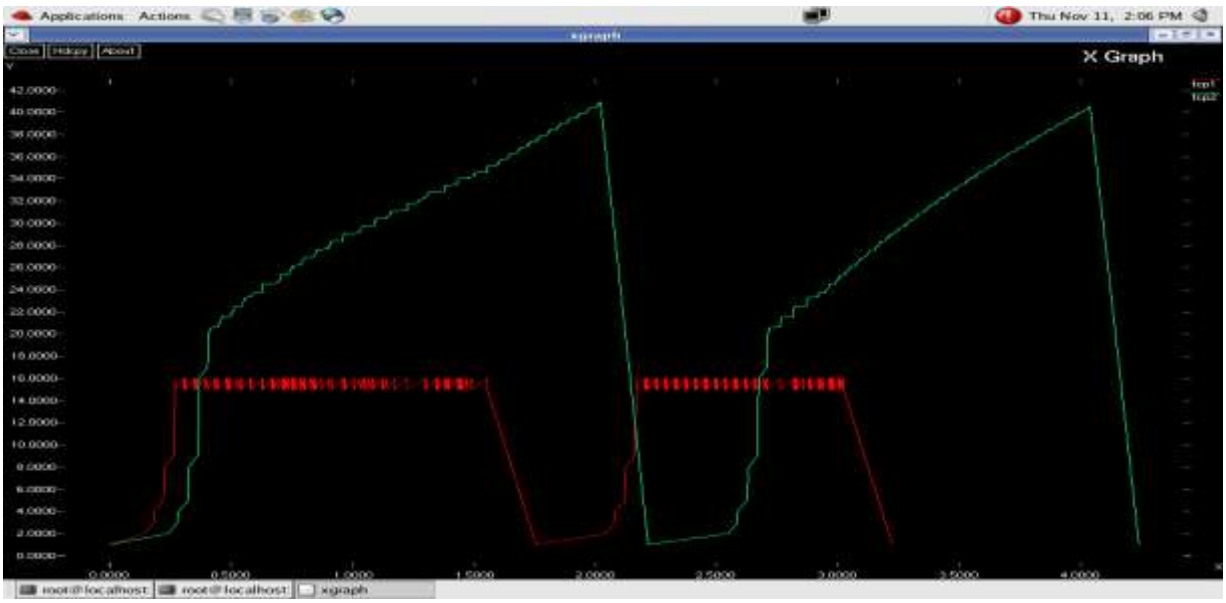
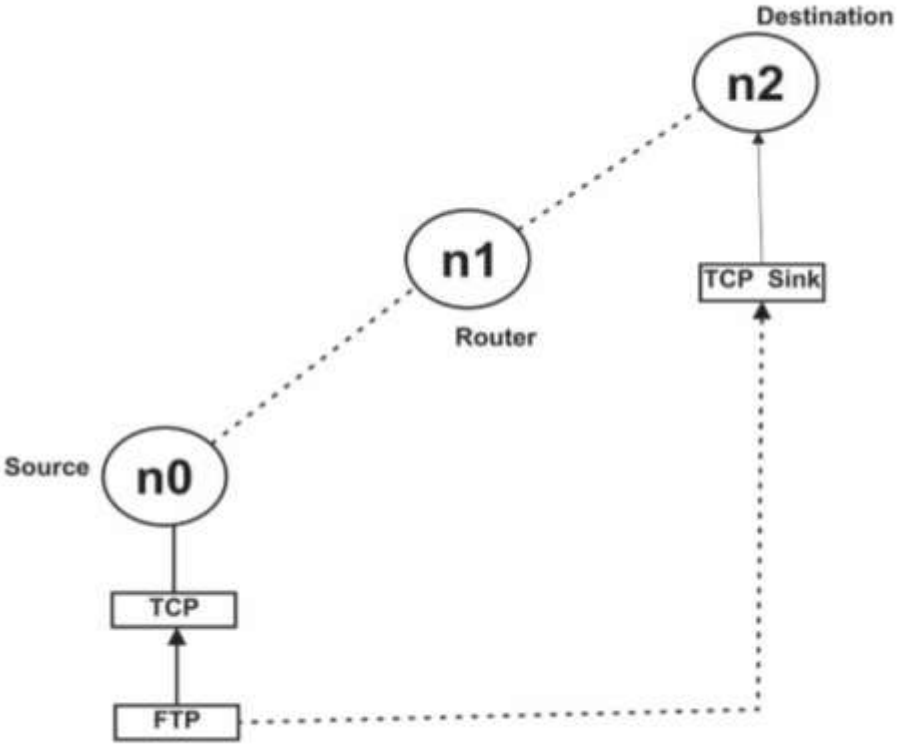


Fig 8: The congestion window graph

EXPERIMENTS-5

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

Design:



Simulation Parameters:

Area	:	1000m *1000m
Simulation Time	:	250 sim sec
Wireless nodes	:	3
Location of nodes	:	As shown in above Figure
Routing Protocol	:	DSDV (Distance Sequenced Distance vector)
Interface queue Type		Queue/DropTail
MAC	:	802_11
Application	:	FTP
Antenna	:	omniantenna

Program:

```
set ns [new Simulator]
set tf [open lab8.tr w]
$ns trace-all $tf
set topo [new Topography]
$topo load_flatgrid 1000 1000
set nf [open lab8.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"

    #The below code is used to give the initial node positions. $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"

#The below code is used to provide the node movements.

$ns at 100 "$n1 **setdest** 550 550 15"

$ns at 190 "$n1 **setdest** 70 70 15"

proc finish {} {

```
    global ns nf tf
    $ns flush-trace
    exec nam lab8.nam &
    close $tf
     exit 0
}
$ns at 250 "finish"
$ns run
```

AWK Script:

```
BEGIN
{
        #include<stdio.h>
        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: %fMbps\n",
((count1*pack1*8)/(time1*1000000)))
Printf("The Throughput from n1 to n2: %f Mbps", ((count2* pack2 * 8)
/(time2*1000000)))
}
```

Output:

ns lab8.tcl

awk –f lab8.awk lab8.tr

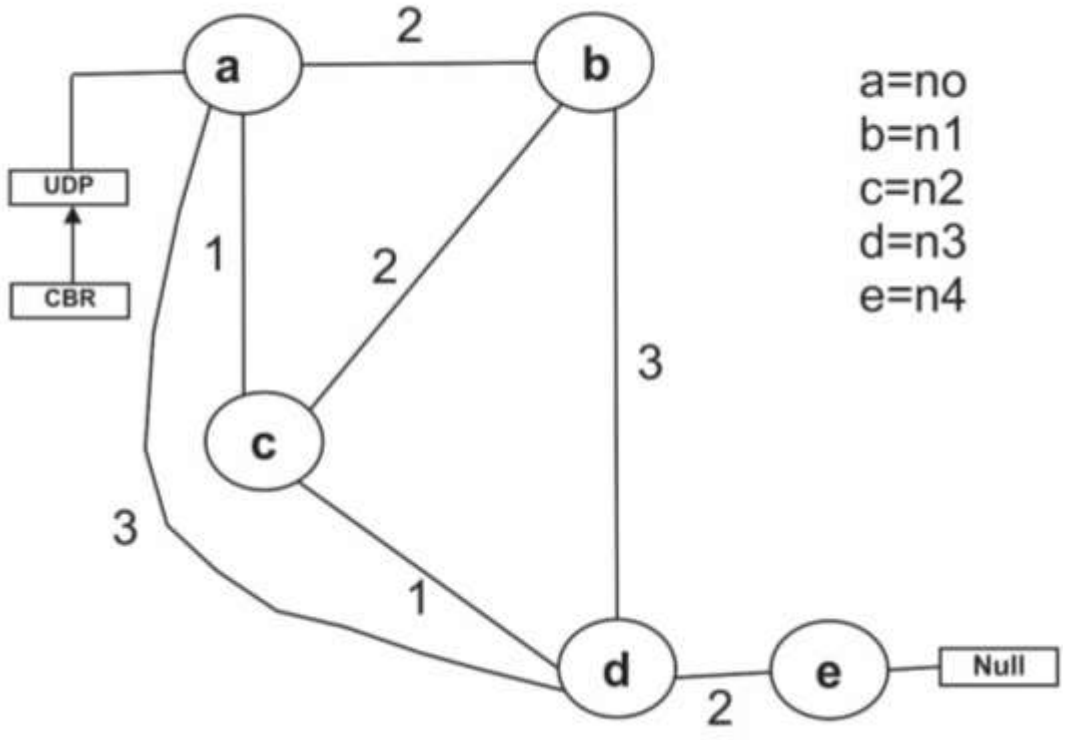
The Throughput from n0 to n1: 5444Mbps

The Throughput from n1 to n2: 345Mbps

EXPERIMENTS-6

Implementation of Link state routing algorithm.

Design:



Program

```
set ns [new Simulator]

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

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

proc finish {} {
        global nf ns tr
        $ns flush-trace
        close $tr
        exec nam out.nam &
        exit 0
        }
set n0 [$ns node]
```

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

$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail

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

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp

set ftp [new Application/FTP]
$ftp attach-agent $tcp

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

set udp [new Agent/UDP]
$ns attach-agent $n2 $udp

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

set null [new Agent/Null]
$ns attach-agent $n3 $null

$ns connect $tcp $sink
$ns connect $udp $null

$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3

$ns rtproto LS

$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"

$ns at 5.0 "finish"

$ns run
```

PART-B

EXPERIMENTS-1

Write a program for

i) Bit stuffing ii) Character stuffing.

i) Bit stuffing

Theory:

Security and Error detection are the most prominent features that are to be provided by any application which transfers data from one end to the other end. One of such a mechanism in tracking errors which may add up to the original data during transfer is known as Stuffing. It is of two types namely Bit Stuffing and the other Character Stuffing. Coming to the Bit Stuffing, 01111110 is appended within the original data while transfer of it. The following program describes how it is stuffed at the sender end and de-stuffed at the receiver end.

Program:

```
#include<string.h>
#include<stdio.h>
#include<conio.h>
void main()
{
	char ch, array[50]={"01111110"},read_array[50];
	int counter=0, i=8,j,k;
	clrscr();
	printf("\n enter the original data stream for bit stuffing\t");
	while ((ch=getche())!='\r')
	{
		if(ch=='1')
			++counter;
		else
			counter=0;
		array[i++]=ch;
		if(counter==5)
		{
			array[i++]='0';
			counter=0;
		}
	}
	strcat(array,"01111110");
	printf("\n the stuffed data stream:\t");
	for(j=0;j<i+8;++j)
		printf("%c",array[j]);
//DESTUFFING
	counter=0;
```

```
        printf("\n The destuffed data stream is: \t");
        for(j=8,k=0;j<i+8;++j)
        {
                if (array[j]=='1')
                        ++counter;
                else
                        counter=0;
                read_array[k++]=array[j];
                if((counter==5)&&(array[j+1]=='0'))
                {
                        j++;
                        counter=0;
                }
        }
        for(j=0;j<k-strlen("01111110");++j)
                printf("%c",read_array[j]);
getch();
}
```

Output:

ii) Character stuffing.

Theory:

Coming to the Character Stuffing, DLESTX and DLEETX are used to denote start and end of character data with some constraints imposed on repetition of charaters as shown in the program below clearly.

Program:

```
#include<stdio.h>
#include<conio.h>
#define DLE 16
#define STX 2
#define ETX 3
#define MAX 100
void main()
{
	char array[100] = {DLE,STX};
	int i=2, j;
	char ch;
	clrscr();
	printf("\n enter the data stream(ctrl+b->STX,ctrl+c->ETX,ctrl+p->DLE);\n");
	while ((ch=getch())!='\r')
	{
		if(ch==DLE)
		{
			array[i++] = DLE;
			printf("DLE");
		}
		else if(ch ==STX)
		{
			printf("STX");
		}
		else if (ch==ETX)
		{
			printf("ETX");
		}
		else
		{
			printf("%c",ch);
		}
		array[i++]=ch;
	}
	array[i++]=DLE;
	array[i++]=ETX;
	printf("\n the character stuffed data stream is:\n");
	for(j=0;j<i;++j)
	{
```

```
                if(array[j]==DLE)
                {
                        printf("DLE");
                }
                else if(array[j]==STX)
                {
                        printf("STX");
                }
                else if(array[j]==ETX)
                {
                        printf("ETX");
                }
                else
                {
                        printf("%c", array[j]);
                }
        }
/* Character Destuffing */
        printf("\n The Character destuffed data stream is: \n");
        for(j=2;j<i-2;++j)
        {
                if(array[j] == DLE)
                {
                        printf("DLE");
                        ++j;
                }
                else if(array[j] == STX)
                {
                        printf("STX");
                }

                else if(array[j] == ETX)
                {
                        printf("ETX");
                }
                else
                {
                        printf("%c",array[j]);
                }
        }
        getch();
}
```

Output:


C:\Turboc2\TC.EXE
enter the number of characters
9
enter the characters
dledleabc
original data
dledleabc
transmitted data:
dlestx
dledledledleabcdleetx
received data:
dledleabc
1.character stuffing
2.exit
enter choice

EXPERIMENTS-2

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

Theory:

Distance Vector routing (DVR) algorithm is unlike Dijkstra's algorithm which is a non-adaptive routing algorithm and means that it is purely static, that is pre-destined and fixed, not flexible in networks where congestions are more prone to occur. DVR is an adaptive routing algorithm in which the information from neighbours is maitained well by each and every node and this helps us to determine the simplest path possible in a changing network. Though, one of the node may fail, still, the destined node is reachable through other possible intermediate nodes that are found out by the DVR algorithm. The perfectly executing program below shows it live below.

Program:

```
#include<stdio.h>

struct node
{
unsigned dist[20];
unsigned from[20];
}rt[10];

int main()
{
int dmat[20][20];
int n,i,j,k,count=0;
clrscr();
printf("\nEnter the number of nodes : ");
scanf("%d",&n);
printf("\nEnter the cost matrix :\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
scanf("%d",&dmat[i][j]);
dmat[i][i]=0;
rt[i].dist[j]=dmat[i][j];
rt[i].from[j]=j;
}
do
{
count=0;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
for(k=0;k<n;k++)
if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])
{
rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];
rt[i].from[j]=k;
count++;
}
}while(count!=0);
```

```
for(i=0;i<n;i++)
{
printf("\n\nState value for router %d is \n",i+1);
for(j=0;j<n;j++)
{
printf("\t\nnode %d via %d Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);
}
}
printf("\n\n");
return 0;
}
```

EXPERIMENTS-3

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

Theory:

Dijkstra's algorithm is a non-adaptive routing algorithm which is very widely used to route packets from source to detination through various routers available during the transmission. It is implemented at the network layer of the architecture where data packets are sent through routers which maitain routing tables that help to denote the exact location to where the destined packets need to be delivered. Major advantage in using Dijkstra's algorithm is that it forwards the data packets from source to destination through the most optimized path in terms of both the distance and cost observed. It prompts the user to enter the number of nodes and the source and destination nodes among them. In addition, the algorithm written below also asks for the neighbours to each node with the distances to reach to them from each node is also prompted. All this data is stored and used further to calculate and estimate the best path possible for data packets to reach their destination from source. Program below explains it in a much better way.

Program:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
      int G[MAX][MAX],i,j,n,u;
      clrscr();
      printf("Enter no. of vertices:");
      scanf("%d",&n);
      printf("\nEnter the adjacency matrix:\n");

      for(i=0;i<n;i++)
            for(j=0;j<n;j++)
                  scanf("%d",&G[i][j]);

      printf("\nEnter the starting node:");
      scanf("%d",&u);
      dijkstra(G,n,u);

      return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

      int cost[MAX][MAX],distance[MAX],pred[MAX];
```

```
int visited[MAX],count,mindistance,nextnode,i,j;

//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
for(i=0;i<n;i++)
      for(j=0;j<n;j++)
            if(G[i][j]==0)
                  cost[i][j]=INFINITY;
            else
                  cost[i][j]=G[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];
                              pred[i]=nextnode;
                        }
      count++;
}

//print the path and distance of each node
for(i=0;i<n;i++)
      if(i!=startnode)
      {
            printf("\nDistance of node%d=%d",i,distance[i]);
            printf("\nPath=%d",i);
```

```
                j=i;
                do
                {
                        j=pred[j];
                        printf("<-%d",j);
                }while(j!=startnode);
        }
}
```

EXPERIMENTS-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

Theory:

CRC means Cyclic Redundancy Check. It is the most famous and traditionally successful mechanism used in error detection through the parity bits installed within the data and obtaining checksum which acts as the verifier to check whether the data retrieved at the receiver end is genuine or not. Various operations are involved in implementing CRC on a data set through CRC generating polynomials. In the program, I have also provided the user to opt for Error detection whereby he can proceed for it. Understand the program below as it is much simpler than pretended to be so.

Program:

```
#include<stdio.h>

#define DEGREE 16
#define MOD2ADD(x,y)(x==y? 0:1)
int result[30];
//Function prototype
void CRC(int length);
int getnext(int*array,int pos);
void main()
{
      int array[30],ch;
      int length,i = 0;
      clrscr();
      printf("Enter the data stream:");
      while((ch = getche()) !='\r')
      {
            array[i++] = ch-'0';
      }
      length = i;
      // Append zeros
      for(i=0;i<DEGREE;i++)
      {
            array[i+length] = 0;
      }
      length+= DEGREE;
      // Duplicate the input data
      for(i=0;i<length;i++)
      {
            result[i] = array[i];
      }
      CRC(length);
      printf("\n The transmitted frame is:");
      for(i=0;i<length-DEGREE;++i)
      {
            printf("%d",array[i]);
      }
```

```
	for(i=length-DEGREE;i<length;++i)
	{
		printf("%d",result[i]);
	}
	printf("\n Enter the stream for which CRC has to be checked:");
	i = 0;
	while((ch = getche()) != '\r')
	{
		array[i++] = ch-'0';
	}
	length = i;

	for(i=0;i<length;i++)
	{
		result[i] = array[i];
	}
	CRC(length);
	printf("\n CHECKSUM:");
	for(i=length-DEGREE;i<length;++i)
	{
		printf("%d",result[i]);
	}
	//getchar()
}
void CRC(int length)
{
	int ccitt[] = {1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};
	int i=0,pos=0,newpos;
	while(pos<length-DEGREE)
	{
		for(i=pos;i<pos+DEGREE+1;++i)
		{
			result[i] = MOD2ADD(result[i],ccitt[i-pos]);
		}
		newpos = getnext(result,pos);
		if(newpos>pos+1)

		{
			pos = newpos-1;
		}
		++pos;
	}
}
int getnext(int*array,int pos)
{
	int i=pos;
	while(array[i] == 0)
	{
		++i;
	}
	return i;
}
```

EXPERIMENTS-5

Implementation of Stop and Wait Protocol and Sliding Window Protocol

i) Program: Stop and Wait Protocol

```
include<stdio.h>
                int timer=0,wait_for_ack=-1,frameQ=0,cansend=1,t=0;
                 main()
                {
                    int i,j,k;
                    int frame[5];
                    //clrscr();
                    printf("enter the time when data frame will be ready\n");
                    for(j=0;j<3;j++)
                    {
                        sender( i, &frame);
                        recv(i);
                    }
                }
                        {
                            wait_for_ack++;
                            if(wait_for_ack==3)
                            {
                        }
                        if(i==frame[t])
                        {
                            frameQ++;
                            t++;
                        }
                        if(frameQ==0)
                            printf("NO FRAME TO SEND at time=%d \n",i);
```

```
                    if(frameQ>0 && cansend==1)
                    {
                        printf("FRAME SEND AT TIME=%d\n",i);
                        cansend=-1;
                        frameQ--;
                        timer++;
                        printf("timer in sender=%d\n",timer);
                    }
                    if(frameQ>0 && cansend==-1)
                        printf("FRAME IN Q FOR TRANSMISSION AT
            TIME=%d\n",i);
                    if(frameQ>0)
        t++;
    printf("frameQ=%d\n",frameQ);
    printf("i=%d    t=%d\n",i,t);
    printf("value in frame=%d\n",frame[t]);
    //    return 0;
    }
    recv(int i )
    {  printf("
timer in recvr=%d\n",timer);
        if(timer>0)
        {
            timer++;
        }
            if(timer==3)
            {
                printf("FRAME ARRIVED AT
TIME=%d\n",i);
```

```
                wait_for_ack=0;
                timer=0;
            }
            else
                printf("
    WAITING FOR FRAME AT TIME %d\n",i);
//                  return 0;
    }
```

ii) Program: Sliding Window Protocol

```
#include<stdio.h>
int main()
{
int w,i,f,frames[50];
printf("Enter window size: ");
scanf("%d",&w);
printf("\nEnter number of frames to transmit: ");
scanf("%d",&f);
printf("\nEnter %d frames: ",f);
for(i=1;i<=f;i++)


d",&frames[i]);
printf("\nWith sliding window protocol the
frames will be sent in the following manner
(assuming no corruption of frames)\n\n");
printf("After sending %d frames at each stage
sender waits for acknowledgement sent by the
receiver\n\n",w);
for(i=1;i<=f;i++)
{
if(i%w==0)
{
printf("%d\n",frames[i]);
printf("Acknowledgement of above frames sent
is received by sender\n\n");
}
else
```

```
printf("%d ",frames[i]);
}
if(f%w!=0)
printf("\nAcknowledgement of above frames sent
is received by sender\n");
return 0;
}
```

EXPERIMENTS-6

Write a program for congestion control using leaky bucket algorithm

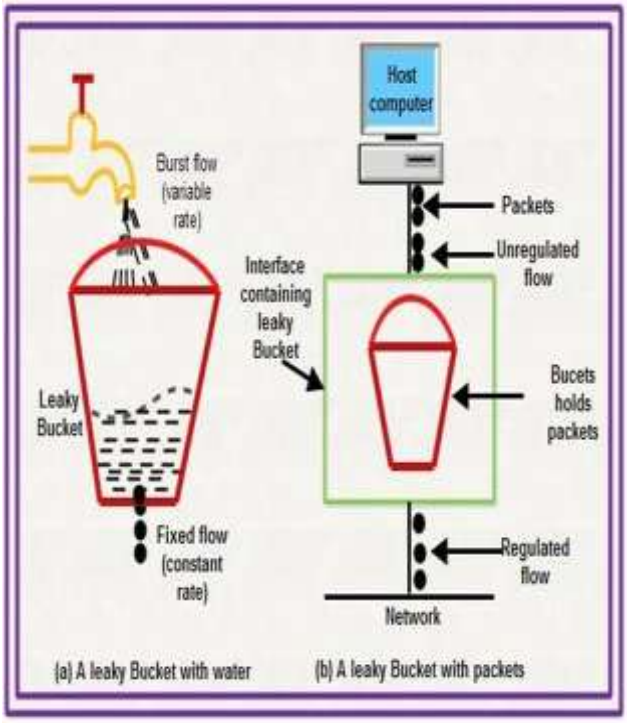
Leaky Bucket Algorithm

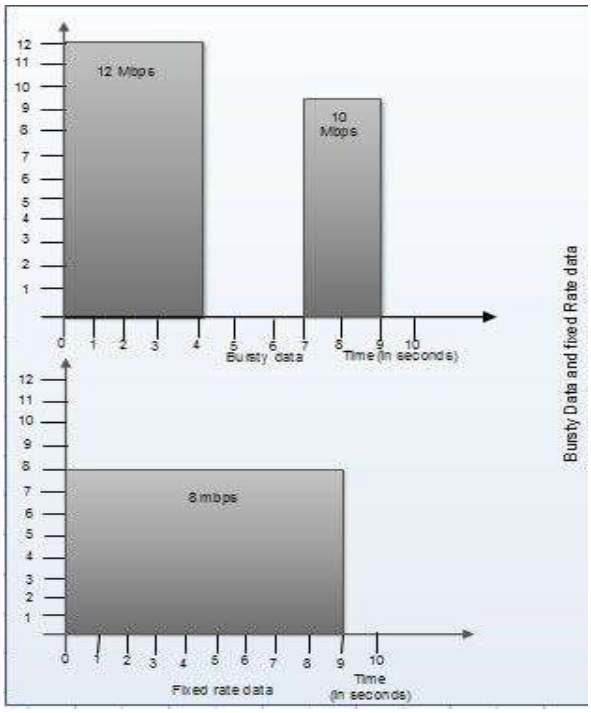
• It is a traffic shaping mechanism that controls the amount and the rate of the traffic sent to the network.

• A leaky bucket algorithm shapes bursty traffic into fixed rate traffic by averaging the data rate.

• Imagine a bucket with a small hole at the bottom.

• The rate at which the water is poured into the bucket is not fixed and can vary but it leaks from the bucket at a constant rate. Thus (as long as water is present in bucket), the rate at which the water leaks does not depend on the rate at which the water is input to the bucket.





• Also, when the bucket is full, any additional water that enters into the bucket spills over the sides and is lost.

• The same concept can be applied to packets in the network. Consider that data is coming from the source at variable speeds. Suppose that a source sends data at 12 Mbps for 4 seconds. Then there is no data for 3 seconds. The source again transmits data at a rate of 10 Mbps for 2 seconds. Thus, in a time span of 9 seconds, 68 Mb data has been transmitted.

If a leaky bucket algorithm is used, the data flow will be 8 Mbps for 9 seconds. Thus constant flow is maintained.

Algorithm:

1. Start
2. Set the bucket size or the buffer size.
3. Set the output rate.
4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size)
6. Stop

Code

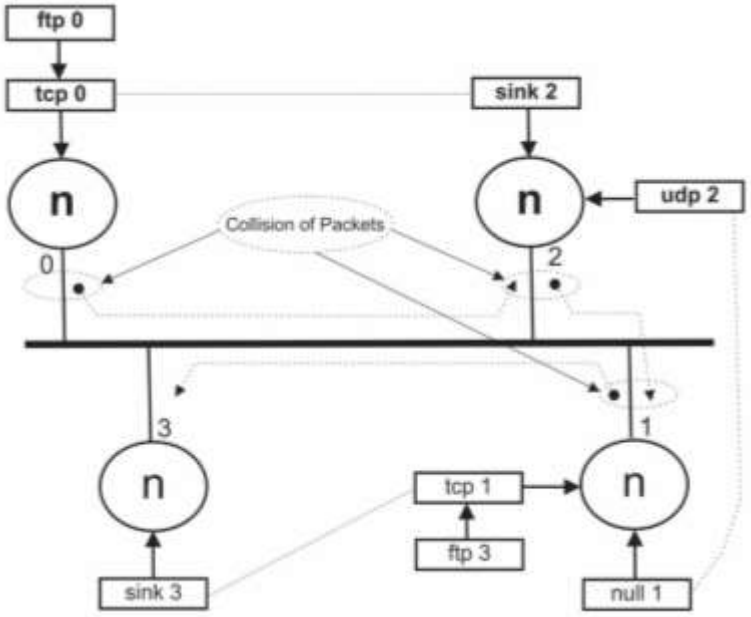
```
#include<stdio.h>
#include<stdlib.h>
#define MIN(x,y) (x>y)?y:x
int main()
{
int orate,drop=0,cap,x,count=0,inp[10]={0},i=0,nsec,ch;
printf("\n enter bucket size : ");
scanf("%d",&cap);
printf("\n enter output rate :");
scanf("%d",&orate);
do
{
printf("\n enter number of packets coming at second %d :",i+1);
scanf("%d",&inp[i]);
i++;
printf("\n enter 1 to contiue 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;
}
```

EXTRA EXPERIMENTS-1

Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and determine collision across different nodes.

Design:



Note:

1. The collision of packets takes place at the nodes n0, n1 and n2. The "c" in the trace file indicating the collision has been occurred at the nodes.

2. The command -trace on is used to see the collision "c" in the trace file as shown below.

$ns make-lan -trace on "$n0 $n1 $n2 $n3" 10Mb 10ms LL

Queue/DropTail Mac/802_3

3. The number of collisions occurred at different nodes are obtained by running the awk script.

Program:

```
set ns [new Simulator]

set tf [open lab6.tr w]

$ns trace-all $tf

set nf [open lab6.nam w]

$ns namtrace-all $nf

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]
```

```
set n3 [$ns node]
set n4 [$ns node]
$ns make-lan -trace on "$n0 $n1 $n2 $n3 $n4" 100mb 10ms LL
Queue/DropTail Mac/802_3
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp0 $sink2
set udp2 [new Agent/UDP]
$ns attach-agent $n2 $udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
set null [new Agent/Null]
$ns attach-agent $n1 $null
$ns connect $udp2 $null
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink3 [new Agent/Null]
$ns attach-agent $n3 $sink3
$ns connect $tcp1 $sink3
$ftp0 set interval_ 0.001
$cbr2 set interval_ 0.001
$ftp1 set interval_ 0.01
proc finish {} {
```

```
global ns nf tf
$ns flush-trace
exec nam lab6.nam &
close $tf
close $nf
exit 0
}
$ns at 0.1 "$cbr2 start"
$ns at 1.2 "$ftp1 start"
$ns at 1.3 "$ftp0 start"
$ns at 5.0 "finish"
$ns run
```

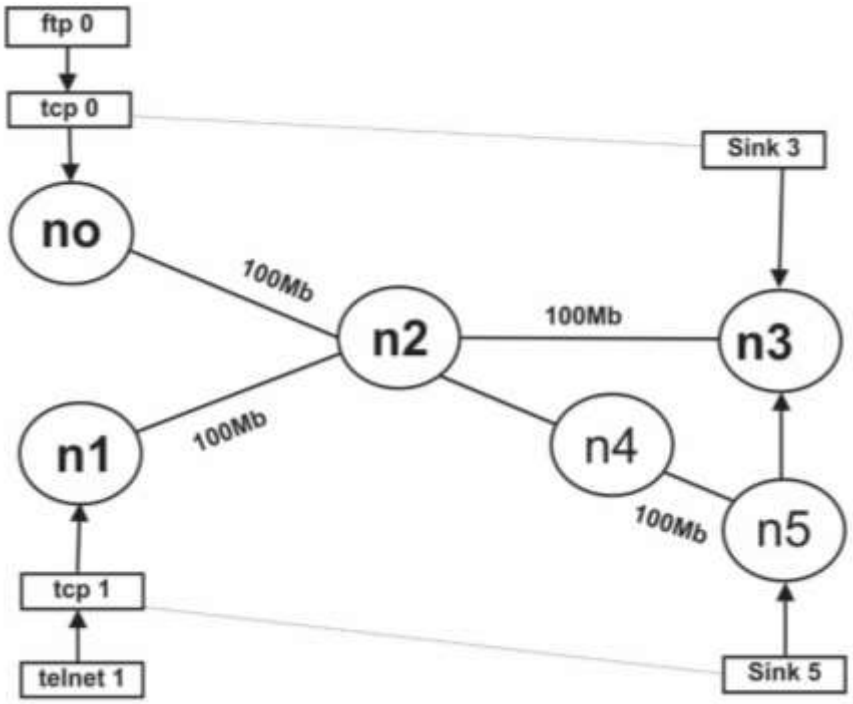
AWK Script:

```
BEGIN{
#include<stdio.h>
count=0
}
{
if($1=="c") {
printf("The [ %s ] Packet occured collision at the
node:[ %s ]\n",$5,$3);
count++
}
}
END{
printf("\n**************************************\n");
printf("\nThe Total Packet Collision %d", count);
printf("\n**************************************\n");
}
```

EXTRA EXPERIMENTS-2

Simulate the different types of internet traffic such as FTP and TELNET over a network and analyze the throughput.

Design:



Note:

1. Change the data rate or the bandwidth of either tcp or the telnet and analyze the throughput

Eg: To vary the data rate of telnet or tcp we use:

$telnet1 set interval_ 0.01 & $ftp0 set interval_ 0.0001

To vary the packet size of telnet or ftp we use

$telnet1 set packetSize_ 5000 & $ftp0 set paceketSize_ 2000

Program:

```
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf
set nf [open lab3.nam w]
$ns namtrace-all $nf
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 label "Source/FTP"
$n1 label "Source/Telnet"
$n3 label "Destination/FTP"
```

```
$n5 label "Desination/Telnet"
$ns color 1 "red"
$ns color 2 "orange"
$ns duplex-link $n0 $n2 100Mb 1ms DropTail
$ns duplex-link $n1 $n2 100Mb 1ms DropTail
$ns duplex-link $n2 $n3 100Mb 1ms DropTail
$ns duplex-link $n2 $n4 100Mb 1ms DropTail
$ns duplex-link $n4 $n5 100Mb 1ms DropTail
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1

set telnet1 [new Application/Telnet]
$telnet1 attach-agent $tcp1
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5
$telnet1 set packetSize_ 1000Mb
$telnet1 set interval_ 0.00001
#The below code is used to connect the tcp agents & sink.
$ns connect $tcp0 $sink3
$ns connect $tcp1 $sink5
#The below code is used to give a color to tcp and telnet
#packets.
$tcp0 set class_ 1
$tcp1 set class_ 2
proc finish { } {
global ns nf tf
exec nam lab3.nam &
close $nf
close $tf
exit 0
}
$ns at 0.1 "$ftp0 start"
$ns at 0.1 "$telnet1 start"
$ns at 15 "finish"
$ns run
```

AWK Script:

```
BEGIN{
#include<stdio.h>
ftppack=0
telpack=0
ftptime=0
teltime=0
```

```
}

{
if($1=="r"&&$3=="2"&&$4=="3")
{
ftppack=ftppack+$6;
ftptime=$2;
}
if($1=="r"&&$3=="4"&&$4=="5")
{
telpack=telpack+$6;
teltime=$2;
}
}
END{
printf("Throughput of Ftp:%fMbps",(ftppack/ftptime)*(8/1000000));
printf("Throughput of tel:%fMbps",(telpack/teltime)*(8/1000000));
}
```

Output:

ns lab3.tcl

awk –f lab3.tcl lab3.tr

Throughput of Ftp:67 Mbps

Throughput of tel:72 Mbps

VIVA QUESTIONS

1. Define Network?
2. What is a Link?
3. What is a node?
4. What is a gateway or Router?
5. What is point-point link?
6. What is Multiple Access?
7. What are the advantages of Distributed Processing?
8. What are the criteria necessary for an effective and efficient network?
9. Name the factors that affect the performance of the network?
10. Name the factors that affect the reliability of the network?
11. Name the factors that affect the security of the network?
12. What are the key elements of protocols?
13. What are the key design issues of a computer Network?
14. Define Bandwidth and Latency?
15. Define Routing?
16. What is a peer-peer process?
17. When a switch is said to be congested?
18. What is semantic gap?
19. What is Round Trip Time?
20. Define the terms Unicasting, Multicasting and Broadcasting?
21. What is Multiplexing?
22. List the layers of OSI
23. Which layers are network support layers?
24. Which layers are user support layers?
25. Which layer links the network support layers and user support layers?
26. What are the concerns of the Physical Layer?
27. What are the responsibilities of Data Link Layer?
28. What are the responsibilities of Network Layer?

29. What are the responsibilities of Session Layer?

30. What are the responsibilities of Presentation Layer?

31. What are the responsibilities of Application Layer?

32. What is CRC?

33. What is Checksum?

34. What are the Data link protocols?

35. Compare Error Detection and Error Correction:

36. What is Framing?

37. What is Fixed Size Framing?

38. Define Character Stuffing?

39. What is Bit Stuffing?

40. What is Flow Control?

41. What is Error Control?

42. What Automatic Repeat Request (ARQ)?

43. What is Stop-and-Wait Protocol?

44. What is Stop-and-Wait Automatic Repeat Request?

45. What is Piggy Backing?

46. What are the two types of transmission technology available?

47. What is subnet?

48. Difference between the communication and transmission.

49. What are the possible ways of data exchange?

50. How Gateway is different from Routers?

51. What is MAC address?

52. Difference between bit rate and baud rate.

53. What is Bandwidth?

54. What are the types of Transmission media?

55. What is Project 802?

56. What are the different type of networking / internetworking devices?

57. What is the minimum and maximum length of the header in the TCP segment and IP datagram?

58. What are major types of networks and explain?

59. What are the important topologies for networks

60. What is mesh network?

61. What is difference between baseband and broadband transmission?

62. What is the difference between routable and non- routable protocols?

63. Why should you care about the OSI Reference Model?

64. What is logical link control?

65. What is multicast routing?