



**NEAR ITPB, CHANNASANDRA, BENGALURU – 560 067**

**Affiliated to VTU, Belagavi**

**Approved by AICTE, New Delhi**

**Recognized by UGC under 2(f) & 12(B)**

**Accredited by NBA & NAAC**

## **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**V SEMESTER**

**COMPUTER NETWORK – MVJ22CG52**

**ACADEMIC YEAR 2024– 2025 (ODD)**

### **LABORATORY MANUAL**

**NAME OF THE STUDENT** : \_\_\_\_\_

**BRANCH** : \_\_\_\_\_

**UNIVERSITY SEAT NO.** : \_\_\_\_\_

**SEMESTER & SECTION** : \_\_\_\_\_

**BATCH** : \_\_\_\_\_

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### VISION:

To create an ambience in excellence and provide innovative emerging programs in Computer Science and Engineering and to bring out future ready engineers equipped with technical expertise and strong ethical values.

### MISSION:

1. **Concepts of computing discipline:** To educate students at undergraduate, postgraduate and doctoral levels in the fundamental and advanced concepts of computing discipline.
2. **Quality Research:** To provide strong theoretical and practical background across the Computer Science and Engineering discipline with the emphasis on computing technologies, quality research, consultancy and trainings.
3. **Continuous Teaching Learning:** To promote a teaching learning process that brings advancements in Computer Science and Engineering discipline leading to new technologies and products.
4. **Social Responsibility and Ethical Values:** To inculcate professional behaviour, innovative research Capabilities, leadership abilities and strong ethical values in the young minds so as to work with the commitment for the betterment of the society.

### Programme Educational Objectives (PEOs):

**PEO01: Current Industry Practices:** Graduates will analyze real world problems and give solution using current industry practices in computing technology.

**PEO02: Research & Higher Studies:** Graduates with strong foundation in mathematics and engineering fundamentals will pursue higher learning, R&D activities and consultancy.

**PEO03: Social Responsibility:** Graduates will be professionals with ethics, who will provide industry growth and social transformation as responsible citizens.

**Programme Outcomes (POs):**

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

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

**Program Specific Outcome:**

1. ***PSO1:Programming:*** Ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, DBMS, and networking for efficient design of computer-based systems of varying complexity.
2. ***PSO2: Practical Solution:*** Ability to practically provide solutions for real world problems with a broad range of programming language and open-source platforms in various computing domains.

**COMPUTER NETWORK LABORATORY**

[As per Choice Based Credit System (CBCS) scheme]

(Academic year 2024 -2025)

**SEMESTER – V**

**Subject Code:** MVJ22CG52

**IA Marks:** 50

**Number of Lecture Hours/Week:** 3

**Exam Marks:** 50

**Total Number of Lecture Hours:** 26

**L : T : P ::** 3 : 0 : 2

**CREDITS – 04**

**Course objective is to:**

This course will enable students to

1. Introduction to fundamental concepts and types of computer networks
2. Demonstrate the TCP/IP and OSI models with merits and demerits.
3. Understand the difference between all communication protocols.

**Prerequisites:**

1. Data Communication
2. Computer Networks

**Course Outcomes (CO's):**

CO No	CO's
CO 1	To familiarize the student with the basic taxonomy and terminology
CO 2	Develop programs related to error detection, CRC-CCITT, distance vector algorithm etc
CO.3	Know how network delivers the packets to destination network
CO 4	Know how switch hopping between mobile towers and Functions of mobile networks
CO 5	Guess the problems in audio/video transfer through network

**Lab Experiments:**

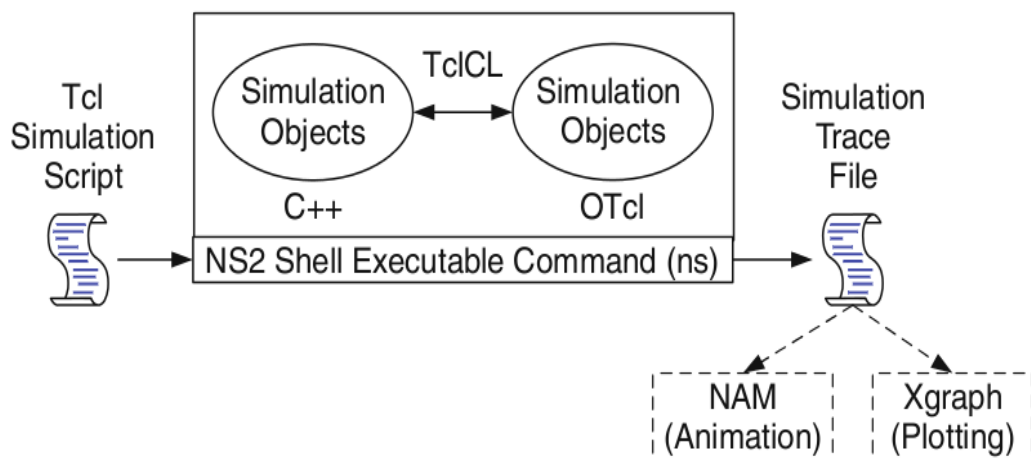
<b>Sl. No.</b>	<b>Programs</b>	<b>RBTL</b>
<b>1</b>	Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer and examine.	<b>L3</b>
<b>2</b>	Write a program for error detecting code using CRC-CCITT (16- bits).	<b>L3</b>
<b>3</b>	Write a program to find the shortest path between vertices using bellman-ford algorithm.	<b>L3</b>
<b>4</b>	Applications using TCP sockets like: a) Echo client and echo server b) Chat c) File Transfer	<b>L3</b>
<b>5</b>	Simulation of DNS using UDP sockets.	<b>L3</b>
<b>6</b>	Write a code for simulating ARP /RARP protocols.	<b>L3</b>
<b>7</b>	Implementation of Stop and Wait Protocol and Sliding Window Protocol.	<b>L3</b>
<b>8</b>	Write a program for congestion control using leaky bucket algorithm.	<b>L3</b>
<b>9</b>	Implement three node point-to-point networks with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.	<b>L3</b>
<b>10</b>	Simulate the transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.	<b>L3</b>
<b>11</b>	Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.	<b>L3</b>
<b>12</b>	Simulate simple ESS and with transmitting nodes in wireless LAN by simulation and determine the performance with respect to transmission of packets.	<b>L3</b>

CO-PO/PSO Mapping															
PO1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-		2	-	-	-	-	-	-	-	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 behaviours.

### 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!
```

#### ○ Variables Command Substitution

```
set a 5  
set len [string length foobar]  
set b $aset len [expr [string length foobar] + 9]
```

#### ○ 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} {inc 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 installation 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 data trace file called “out.tr” and a nam visualization trace file called “out.nam”. Within the tclscript, 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 namout.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,

```
$ns at 125.0 “finish”
```

will be used to call “**finish**” at time 125sec.Indeed,the **at** method of the simulator allows us to schedule events explicitly.

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 created which 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) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).

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

The below 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.

```
set cbr [new
Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set packetSize_ 100

$cbr set rate_ 0.01Mb

$cbr set random_ false
```

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 filter input and select lines for the action component to act upon. The selection criteria are 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 their choice. You can now print a serial number, using the variable `kount`, and apply it to 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 hold large awk programs in separate files 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 is 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 be 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"}'emp.lst
```



**Experiment No:1** Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute.**AIM:**

To Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute ping.

**PRE LAB DISCUSSION:****Tcpdump:**

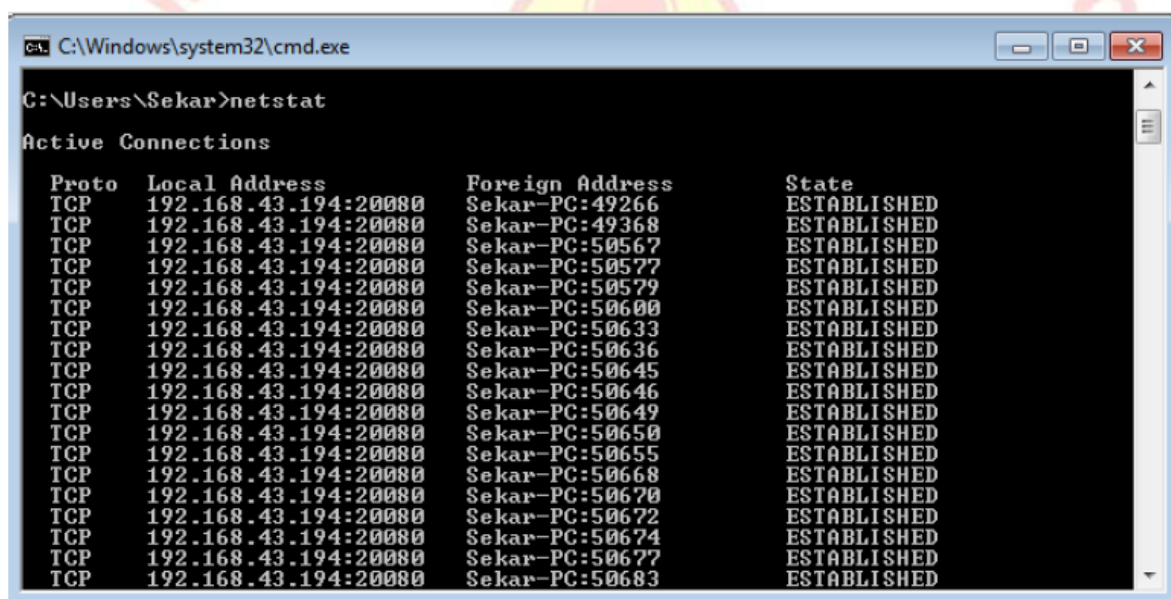
The tcpdump utility allows you to capture packets that flow within your network to assist in network troubleshooting. The following are several examples of using tcpdump with different options. Traffic is captured based on a specified filter.

**Netstat**

Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems.

Netstat provides information and statistics about protocols in use and current TCP/IP network connections.

#netstat



```
C:\Windows\system32\cmd.exe

C:\Users\Sekar>netstat

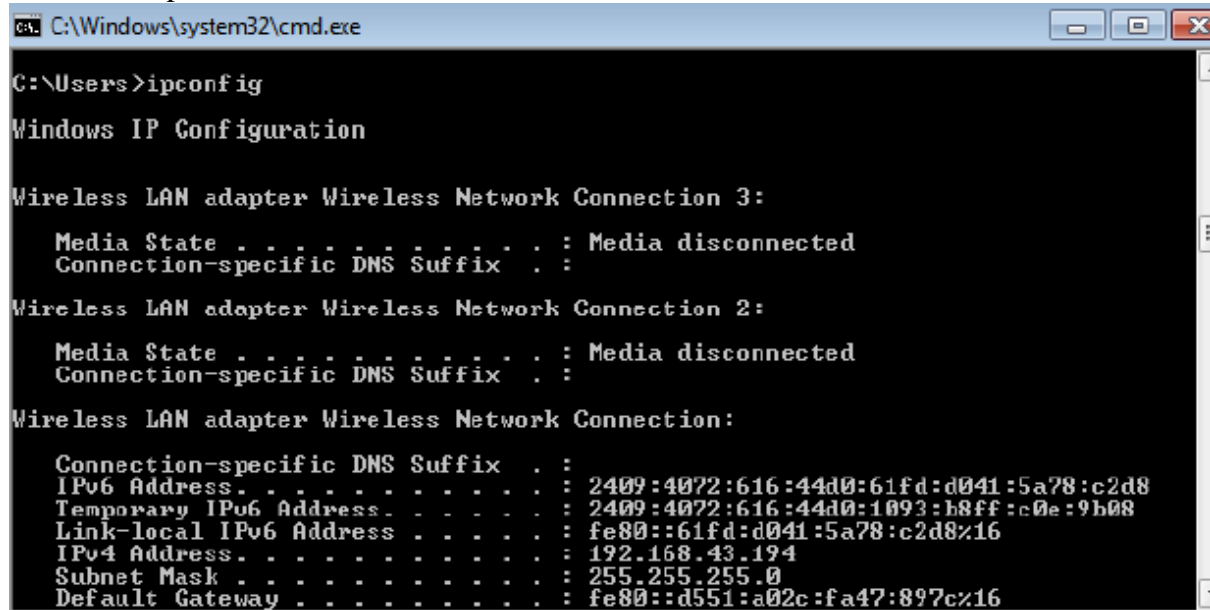
Active Connections

Proto Local Address Foreign Address State
TCP 192.168.43.194:20080 Sekar-PC:49266 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:49368 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50567 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50577 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50579 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50600 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50633 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50636 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50645 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50646 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50649 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50650 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50655 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50668 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50670 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50672 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50674 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50677 ESTABLISHED
TCP 192.168.43.194:20080 Sekar-PC:50683 ESTABLISHED
```

**Ipconfig**

**ipconfig** is a console application designed to run from the Windows command prompt. This utility allows you to get the IP address information of a Windows computer.

From the command prompt, type **ipconfig** to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter.



```

C:\Windows\system32\cmd.exe

C:\Users>ipconfig

Windows IP Configuration

Wireless LAN adapter Wireless Network Connection 3:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Wireless Network Connection 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter Wireless Network Connection:

    Connection-specific DNS Suffix  . :
    IPv6 Address. . . . . : 2409:4072:616:44d0:61fd:d041:5a78:c2d8
    Temporary IPv6 Address. . . . . : 2409:4072:616:44d0:1093:b8ff:c0e:9b08
    Link-local IPv6 Address . . . . . : fe80::61fd:d041:5a78:c2d8%16
    IPv4 Address. . . . . : 192.168.43.194
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : fe80::d551:a02c:fa47:897c%16
  
```

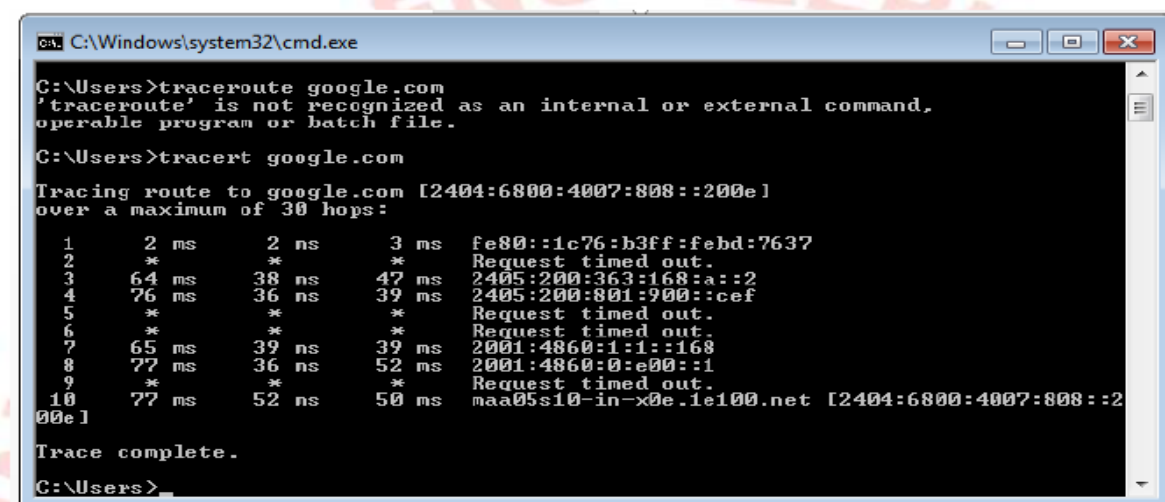
## Nslookup

The **nslookup**(which stands for *name server lookup*) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System.

### Trace route:

Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute also records the time taken for each hop the packet makes during its route to the destination

```
#tracert google.com
```



```

C:\Windows\system32\cmd.exe

C:\Users>tracert google.com
'tracert' is not recognized as an internal or external command,
operable program or batch file.

C:\Users>tracert google.com

Tracing route to google.com [2404:6800:4007:808::200e1]
over a maximum of 30 hops:
  0  1  2 ms  2 ns  3 ms  fe80::1c76:b3ff:febd:7637
  2  *  *  *  *  Request timed out.
  3  64 ms  38 ns  47 ms  2405:200:363:168:a::2
  4  76 ms  36 ns  39 ms  2405:200:801:900::cef
  5  *  *  *  *  Request timed out.
  6  *  *  *  *  Request timed out.
  7  65 ms  39 ns  39 ms  2001:4860:1:1::168
  8  77 ms  36 ns  52 ms  2001:4860:0:e00::1
  9  *  *  *  *  Request timed out.
 10  77 ms  52 ns  50 ms  maa05s10-in-x0e.1e100.net [2404:6800:4007:808::200e1]

Trace complete.

C:\Users>
  
```

## Commands:

Tcpdump:

### Display traffic between 2 hosts:

To display all traffic between two hosts (represented by variables host1 and host2): # tcpdump host host1 and host2

### Display traffic from a source or destination host only:

To display traffic from only a source (src) or destination (dst) host:

# tcpdumpsrc host

# tcpdumpdst host

### Display traffic for a specific protocol

Provide the protocol as an argument to display only traffic for a specific protocol, for example tcp, udp, icmp, arp

tcpdump protocol

For example to display traffic only for the tcp traffic :

# tcpdumptcp

### Filtering based on source or destination port

To filter based on a source or destination port:

# tcpdumpsrc port ftp

# tcpdumpdst port http

## 2.Netstat

Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems.

Netstat provides information and statistics about protocols in use and current TCP/IP network connections. The Windows help screen (analogous to a Linux or UNIX for netstat reads as follows:

Displays protocol statistics and current TCP/IP network connections.

#netstat

## 3. ipconfig

In Windows, **ipconfig** is a console application designed to run from the Windows command prompt. This utility allows you to get the IP address information of a Windows computer.

### Using ipconfig

From the command prompt, type **ipconfig** to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter.

#ipconfig

## 4.nslookup

The **nslookup**(which stands for *name server lookup*) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System.

The nslookup command is a powerful tool for diagnosing DNS problems. You know you're experiencing a DNS problem when you can access a resource by specifying its IP address but not its DNS name.

#nslookup

## 5.Trace route:

Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live

(TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop.

Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute also records the time taken for each hop the packet makes during its route to the destination.

Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values.

The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop. Traceroute sends packets with TTL values that gradually increase from packet to packet, starting with TTL value of one. Routers decrement TTL values of packets by one when routing and discard packets whose TTL value has reached zero, returning the ICMP error message ICMP Time Exceeded.

### **Experiment No: 2**

Write a program for error detecting code using CRC-CCITT (16- bits).

AIM: To Implement a program for error detecting code using CRC-CCITT (16- bits).

Program

```
import java.util.Scanner;
import java.io.*;
public class CRC1 {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        //Input Data Stream
        System.out.print("Enter message bits: ");
        String message = sc.nextLine();
        System.out.print("Enter generator: ");
        String generator = sc.nextLine();
        int data[] = new int[message.length() + generator.length() - 1];
        int divisor[] = new int[generator.length()];
        for(int i=0;i<message.length();i++)
            data[i] = Integer.parseInt(message.charAt(i)+"");
        for(int i=0;i<generator.length();i++)
            divisor[i] = Integer.parseInt(generator.charAt(i)+"");

        //Calculation of CRC
        for(int i=0;i<message.length();i++)
        {
```

```
if(data[i]==1)
for(int j=0;j<divisor.length;j++)
data[i+j] ^= divisor[j];
}

//Display CRC
System.out.print("The checksum code is: ");
for(int i=0;i<message.length();i++)
data[i] = Integer.parseInt(message.charAt(i)+"");
for(int i=0;i<data.length;i++)
System.out.print(data[i]);
System.out.println();

//Check for input CRC code
System.out.print("Enter checksum code: ");
message = sc.nextLine();
System.out.print("Enter generator: ");
generator = sc.nextLine();
data = new int[message.length() + generator.length() - 1];
divisor = new int[generator.length()];
for(int i=0;i<message.length();i++)
data[i] = Integer.parseInt(message.charAt(i)+"");
for(int i=0;i<generator.length();i++)
divisor[i] = Integer.parseInt(generator.charAt(i)+"");

//Calculation of remainder
for(int i=0;i<message.length();i++) {
if(data[i]==1)
for(int j=0;j<divisor.length;j++)
data[i+j] ^= divisor[j];
}

//Display validity of data
boolean valid = true;
for(int i=0;i<data.length;i++)
if(data[i]==1){
valid = false;
break;
}

if(valid==true)
System.out.println("Data stream is valid");
else
System.out.println("Data stream is invalid. CRC error occurred.");
```

}

}

**/\*Output of CRC cyclic redundancy check program:**

Enter no. of data bits : 14

Enter no. of generator bits : 4

Enter data bits : 1 1 0 1 0 0 1 1 1 0 1 1 0 0

Enter generator bits : 1 0 1 1

The code bits added are : 100

The code data is : 11010011101100100

\*/

### **Experiment No:3**

**Aim:** Write a program to find the shortest path between vertices using bellman-ford algorithm.

This is a java program to find shortest path from a single vertex. The Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph.[1] It is slower than Dijkstra’s algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers.

Here is the source code of the Java Program to Use the Bellman-Ford Algorithm to Find the Shortest Path Between Two Vertices Assuming that Negative Size Edges Exist in the Graph. The Java program is successfully compiled and run on a Windows system. The program output is also shown below.

**Program**

```
import java.util.Scanner;
public class ford
{
    private int D[];
    private int num_ver;
    public static final int MAX_VALUE = 999;
    public ford(int num_ver)
    {
this.num_ver = num_ver;
        D = new int[num_ver + 1];
    }

    public void BellmanFordEvaluation(int source, int A[][])
    {
        for (int node = 1; node <= num_ver; node++)
        {
            D[node] = MAX_VALUE;
        }

        D[source] = 0;

        for (int node = 1; node <= num_ver - 1; node++)
        {
            for (int sn = 1; sn <= num_ver; sn++)
            {
                for (int dn = 1; dn <= num_ver; dn++)
                {
                    if (A[sn][dn] != MAX_VALUE)
                    {
                        if (D[dn] > D[sn]+ A[sn][dn])
                            D[dn] = D[sn] + A[sn][dn];
                    }
                }
            }
        }
    }
}
```

```
    }

    for (int sn = 1; sn<= num_ver; sn++)
    {
        for (int dn = 1; dn<= num_ver; dn++)
        {
            if (A[sn][dn] != MAX_VALUE)
            {
                if (D[dn] > D[sn]+ A[sn][dn])
                System.out.println("The Graph contains negative egde cycle");
            }
        }
    }
    for (int vertex = 1; vertex <= num_ver; vertex++)
    {
        System.out.println("distance of source"+source+"to"+vertex+"is" + D[vertex]);
    }
}

public static void main(String[ ] args)
{
    int num_ver = 0;
    int source;
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the number of vertices");
    num_ver = scanner.nextInt();

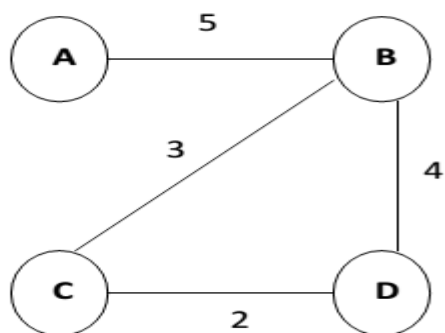
    int A[][] = new int[num_ver + 1][num_ver + 1];
    System.out.println("Enter the adjacency matrix");
    for (int sn = 1; sn<= num_ver; sn++)
    {
        for (int dn = 1; dn<= num_ver; dn++)
        {
            A[sn][dn] = scanner.nextInt();
            if (sn == dn)
            {
                A[sn][dn] = 0;
                continue;
            }
            if (A[sn][dn] == 0)
            {
                A[sn][dn] = MAX_VALUE;
            }
        }
    }
}
```



```
}
```

```
System.out.println("Enter the source vertex");
    source = scanner.nextInt();
    ford b = new ford (num_ver);
    b.BellmanFordEvaluation(source, A);
    scanner.close();
}
```

**Input graph:**



Output:

[root@localhost ~]# vi BellmanFord.java

```
root@localhost: ~
File Edit View Terminal Help
[root@localhost ~]# javac BellmanFord.java
[root@localhost ~]# java BellmanFord
Enter the number of vertices
4
Enter the adjacency matrix
0 5 0 0
5 0 3 4
0 3 0 2
0 4 2 0
Enter the source vertex
2
distance of source 2 to 1 is 5
distance of source 2 to 2 is 0
distance of source 2 to 3 is 3
distance of source 2 to 4 is 4
[root@localhost ~]#
```

## **Experiment No 4**

Applications using TCP sockets like:

- a) Echo client and echo server
- b) Chat
- c) File Transfer

### **Echo client and echo server**

#### **ALGORITHM**

##### **Client**

1. Start
2. Create the TCP socket
3. Establish connection with the server
4. Get the message to be echoed from the user
5. Send the message to the server
6. Receive the message echoed by the server
7. Display the message received from the server
8. Terminate the connection
9. Stop

##### **Server**

1. Start
2. Create TCP socket, make it a listening socket
3. Accept the connection request sent by the client for connection establishment
4. Receive the message sent by the client
5. Display the received message
6. Send the received message to the client from which it receives
7. Close the connection when client initiates termination and server becomes a listening server, waiting for clients.
8. Stop.

```
import java.net.*;
import java.io.*;
public class EServer
{
    public static void main(String args[])
    {
        ServerSocket s=null;
        String line;
        DataInputStream is;
        PrintStream ps;
        Socket c=null;
        try
        {
            s=new ServerSocket(9000);
        }
        catch(IOException e)
```

```
{
System.out.println(e);
}
try
{
c=s.accept();
is=new DataInputStream(c.getInputStream());
ps=new PrintStream(c.getOutputStream());
while(true)
{
line=is.readLine();
ps.println(line);
}
}
catch(IOException e)
{
System.out.println(e);
}
}
}
```

### **EClient.java**

```
import java.net.*;
import java.io.*;
public class EClient
{
public static void main(String arg[])
{
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
try
{
InetAddress ia = InetAddress.getLocalHost();
c=new Socket(ia,9000);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
while(true)
{
```

```
System.out.println("Client:");
line=is.readLine();
os.println(line);
System.out.println("Server:" + is1.readLine());
}}
catch(IOException e)
{
System.out.println("Socket Closed!");

}
}
```

## OUTPUT

### Server

```
C:\Program Files\Java\jdk1.5.0\bin>javac EServer.java
C:\Program Files\Java\jdk1.5.0\bin>java EServer
C:\Program Files\Java\jdk1.5.0\bin>
```

### Client

```
C:\Program Files\Java\jdk1.5.0\bin>javac EClient.java
C:\Program Files\Java\jdk1.5.0\bin>java EClient
Client: Hai Server
Server: Hai Server
Client: Hello
Server: Hello
Client: end
Server: end
Client: ds
Socket Closed!
```

## 4b. Chat

### ALGORITHM

#### Client

1. Start
2. Create the UDP datagram socket
3. Get the request message to be sent from the user
4. Send the request message to the server
5. If the request message is "END" go to step 10
6. Wait for the reply message from the server
7. Receive the reply message sent by the server
8. Display the reply message received from the server
9. Repeat the steps from 3 to 8
10. Stop

#### Server

1. Start

2. Create UDP datagram socket, make it a listening socket
3. Receive the request message sent by the client
4. If the received message is "END" go to step 10
5. Retrieve the client's IP address from the request message received
6. Display the received message
7. Get the reply message from the user
8. Send the reply message to the client
9. Repeat the steps from 3 to 8.
10. Stop.

### **SERVER:**

```
import java.net.*;
```

```
import java.io.*;
```

```
import java.util.*;
```

```
public class UDPServer
```

```
{
```

```
public static void main(String args[])throws SocketException,IOException
```

```
{
```

```
DatagramSocket serversocket=new DatagramSocket(9902);
```

```
Scanner sc=new Scanner(System.in);
```

```
while(true)
```

```
{
```

```
byte[] receivebuffer=new byte[1024];
```

```
byte[] sendbuffer=new byte[1024];
```

```
DatagramPacket recvpkt=new DatagramPacket(receivebuffer,receivebuffer.length);
```

```
serversocket.receive(recvpkt);
```

```
InetAddress ip=recvpkt.getAddress();
```

```
int portno=recvpkt.getPort();
```

```
String clientdata= new String(recvpkt.getData());
```

```
System.out.println("\n Client: "+clientdata);
```

```
System.out.println("\n Server:");
```

```
String serverdata=sc.nextLine();
sendbuffer=serverdata.getBytes();
DatagramPacket sendpacket=new DatagramPacket(sendbuffer,sendbuffer.length,ip,portno);
serversocket.send(sendpacket);
if(serverdata.equalsIgnoreCase("bye"))
{
System.out.println("Connected");
break;
}
}
serversocket.close();
}
```

### **CLIENT :**

```
import java.io.*;
import java.util.*;
import java.net.*;

public class UDPClient
{
public static void main(String args[])throws SocketException,IOException
{
Scanner sc=new Scanner(System.in);
InetAddress ip=InetAddress.getByName("localhost");
DatagramSocket clientsocket=new DatagramSocket();
while(true)
{
byte[] sendbuffer=new byte[1024];
byte[] receivebuffer=new byte[1024];
System.out.println("\n Client:");
```

```
String clientdata=sc.nextLine();
sendbuffer=clientdata.getBytes();
DatagramPacket sendpacket=new DatagramPacket(sendbuffer,sendbuffer.length,ip,9902);
clientsocket.send(sendpacket);
if(clientdata.equalsIgnoreCase("bye"))
{
    System.out.println("connection end by client");
    break;
}
DatagramPacket receivepacket=new DatagramPacket(receivebuffer,receivebuffer.length);
clientsocket.receive(receivepacket);
String serverdata=new String(receivepacket.getData());
System.out.println("\n Server:"+serverdata);
}
clientsocket.close();
}
}
```

### **Server**

```
C:\Program Files\Java\jdk1.5.0\bin>javac UDPserver.java
C:\Program Files\Java\jdk1.5.0\bin>java UDPserver
press ctrl+c to quit the program
Client: Hai Server
Server: Hello Client
Client: How are You
Server: I am Fine
```

### **Client**

```
C:\Program Files\Java\jdk1.5.0\bin>javac UDPclient.java
C:\Program Files\Java\jdk1.5.0\bin>java UDPclient
server waiting
Client: Hai Server
Server: Hello Clie
Client: How are You
Server: I am Fine
Client: end
```

#### 4c. File Transfer

##### AIM:

To write a java program for file transfer using TCP Sockets.

##### Algorithm

###### Server

1. Import java packages and create class file server.
2. Create a new server socket and bind it to the port.
3. Accept the client connection
4. Get the file name and stored into the BufferedReader.
5. Create a new object class file and realine.
6. If file is exists then FileReader read the content until EOF is reached.
7. Stop the program.

###### Client

1. Import java packages and create class file server.
2. Create a new server socket and bind it to the port.
3. Now connection is established.
4. The object of a BufferedReader class is used for storing data content which has been retrieved from socket object.
5. The connection is closed.
6. Stop the program.

##### File Server :

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket
public class FileServer
{
    public static void main(String[] args) throws Exception
    {
        //Initialize Sockets
        ServerSocket ssock = new ServerSocket(5000); Socket
        socket = ssock.accept();
        //The InetAddress specification
        InetAddress IA = InetAddress.getByName("localhost");
        //Specify the file
        File file = new File("e:\\Bookmarks.html");
        FileInputStream fis = new FileInputStream(file);
        BufferedInputStream bis = new BufferedInputStream(fis);

        OutputStream os = socket.getOutputStream(); //Read
        File Contents into contents array
        byte[] contents;
        long fileLength = file.length();
```



```
long current = 0;
long start = System.nanoTime();
while(current!=fileLength){
int size = 10000;
if(fileLength - current >= size)
current += size;
else{
size = (int)(fileLength - current);
current = fileLength;
}
contents = new byte[size];

bis.read(contents, 0, size);
os.write(contents);
System.out.print("Sending file ... "+(current*100)/fileLength+"% complete!");
}
os.flush();
//File transfer done. Close the socket connection!
socket.close();
sock.close();
System.out.println("File sent succesfully!");
} }
```

**File Client:**

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.Socket;
public class FileClient {
public static void main(String[] args) throws Exception{
//Initialize socket
Socket socket = new Socket(InetAddress.getByName("localhost"), 5000); byte[]
contents = new byte[10000];
//Initialize the FileOutputStream to the output file's full path. FileOutputStreamfos = new
FileOutputStream("e:\\Bookmarks1.html");
BufferedOutputStreambos = new BufferedOutputStream(fos);
InputStream is = socket.getInputStream();
//No of bytes read in one read() call
int bytesRead = 0;
while((bytesRead=is.read(contents))!=-1)
bos.write(contents, 0, bytesRead);

bos.flush();
socket.close();
System.out.println("File saved successfully!");
}

}
```

### Output

#### server

```
E:\nwlab>java FileServer
Sending file ... 9% complete!
Sending file ... 19% complete!
Sending file ... 28% complete!
Sending file ... 38% complete!
Sending file ... 47% complete!
Sending file ... 57% complete!
Sending file ... 66% complete!
Sending file ... 76% complete!
Sending file ... 86% complete!
Sending file ... 95% complete!
Sending file ... 100% complete!
File sent successfully!
```

#### E:\nwlab>client

```
E:\nwlab>java FileClient
File saved successfully!
E:\nwlab>
```

### Experiment No 5

Simulation of DNS using UDP sockets.

**AIM :**To write a java program for DNS application

#### **PRE LAB DISCUSSION:**

The Domain Name System (DNS) is a hierarchical decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities.

The domain name space refers a hierarchy in the internet naming structure. This hierarchy has multiple levels (from 0 to 127), with a root at the top. The following diagram shows the domain name space hierarchy.

Name server contains the DNS database. This database comprises of various names and their corresponding IP addresses. Since it is not possible for a single server to maintain entire DNS database, therefore, the information is distributed among many DNS servers. Types of Name Servers  
Root Server is the top level server which consists of the entire DNS tree. It does not contain the information about domains but delegates the authority to the other server Primary Server stores a file about its zone. It has authority to create, maintain, and update the zone file. Secondary Server transfers complete information about a zone from another server which may be primary or secondary server. The secondary server does not have authority to create or update a zone file.

DNS is a TCP/IP protocol used on different platforms. The domain name space is divided into three different sections: generic domains, country domains, and inverse domain.

The main function of DNS is to translate domain names into IP Addresses, which computers can understand. It also provides a list of mail servers which accept Emails for each domain name. Each domain name in DNS will nominate a set of name servers to be authoritative for its DNS records

### ALGORITHM

#### Server

1. Start
2. Create UDP datagram socket
3. Create a table that maps host name and IP address
4. Receive the host name from the client
5. Retrieve the client's IP address from the received datagram
6. Get the IP address mapped for the host name from the table.
7. Display the host name and corresponding IP address
8. Send the IP address for the requested host name to the client
9. Stop.

#### Client

1. Start
2. Create UDP datagram socket.
3. Get the host name from the client
4. Send the host name to the server
5. Wait for the reply from the server
6. Receive the reply datagram and read the IP address for the requested host name
7. Display the IP address.
8. Stop

### Simulation of DNS using UDP Sockets

#### DNS Server

```
java import java.io.*;
import java.net.*;
public class udpdnserver
{
private static int indexOf(String[] array, String str)
{
str = str.trim();
for (int i=0; i<array.length; i++)
{
if (array[i].equals(str))
return i;
}
return -1;
}
public static void main(String arg[])throws IOException
{
String[] hosts = {"yahoo.com", "gmail.com", "cricinfo.com", "facebook.com"};
String[] ip = {"68.180.206.184", "209.85.148.19", "80.168.92.140", "69.63.189.16"};
System.out.println("Press Ctrl + C to Quit");
while (true)
{
DatagramSocketserversocket=new DatagramSocket(1362);
```

```
byte[] senddata = new byte[1021]; byte[] receivedata = new byte[1021];
DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);
serversocket.receive(recvpack);
String sen = new String(recvpack.getData());

InetAddress ipAddress = recvpack.getAddress();
int port = recvpack.getPort();
String capsent;
System.out.println("Request for host " + sen);
if(indexOf (hosts, sen) != -1)
capsent = ip[indexOf (hosts, sen)];
else
capsent = "Host Not Found";
senddata = capsent.getBytes();
DatagramPacket pack = new DatagramPacket (senddata, senddata.length,ipaddress,port);
serversocket.send(pack);
serversocket.close();
}}}
```

### **UDP DNS Client**

```
java import java.io.*;
import java.net.*;
public class udpdnsclient
{
public static void main(String args[])throws IOException
{
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
DatagramSocket clientsocket = new DatagramSocket();
InetAddress ipAddress;
if (args.length == 0)
ipAddress = InetAddress.getLocalHost();
else
ipAddress = InetAddress.getByName(args[0]);
byte[] senddata = new byte[1024];
byte[] receivedata = new byte[1024];
int portaddr = 1362;
System.out.print("Enter the hostname : ");
String sentence = br.readLine();
Senddata = sentence.getBytes();
DatagramPacket pack = new DatagramPacket(senddata,senddata.length, ipAddress,portaddr);
clientsocket.send(pack);
DatagramPacket recvpack =new DatagramPacket(receivedata,receivedata.length);
clientsocket.receive(recvpack);
String modified = new String(recvpack.getData());
System.out.println("IP Address: " + modified);
clientsocket.close();
}}
```

## OUTPUT

### Server

```
javac udpdnsserver.java
java udpdnsserver
Press Ctrl + C to Quit Request for host yahoo.com
Request for host cricinfo.com
Request for host youtube.com
```

### Client

```
>javac udpdnscient.java
>java udpdnscient
Enter the hostname : yahoo.com
IP Address: 68.180.206.184
>java udpdnscient
Enter the hostname : cricinfo.com
IP Address: 80.168.92.140
>java udpdnscient
Enter the hostname : youtube.com
IP Address: Host Not Found
```

## Experiment No 6

Write a code for simulating ARP /RARP protocols.

### AIM:

To write a java program for simulating ARP and RARP protocols using TCP.

### PRE LAB DISCUSSION:

Address Resolution Protocol (ARP) is a low-level network protocol for translating network layer addresses into link layer addresses. ARP lies between layers 2 and 3 of the OSI model, although ARP was not included in the OSI framework and allows computers to introduce each other across a network prior to communication. Because protocols are basic network communication units, address resolution is dependent on protocols such as ARP, which is the only reliable method of handling required tasks.

The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given internet layer address,

When configuring a new network computer, each system is assigned an Internet Protocol (IP) address for primary identification and communication. A computer also has a unique media access control (MAC) address identity. Manufacturers embed the MAC address in the local area network (LAN) card. The MAC address is also known as the computer's physical address.

Address Resolution Protocol (ARP) is used to resolve an IPv4 address (32 bit Logical Address) to the physical address (48 bit MAC Address). Network Applications at the Application Layer use IPv4 Address to communicate with another device.

Reverse Address Resolution Protocol (RARP) is a network protocol used to resolve a data link layer address to the corresponding network layer address. For example, RARP is used to resolve a Ethernet MAC address to an IP address.

The client broadcasts a RARP packet with an ethernet broadcast address, and it's own physical address in the data portion. The server responds by telling the client it's IP address. Note there is no name sent. Also note there is no security.

Media Access Control (MAC) addresses need to be individually configured on the servers by an administrator. RARP is limited to serving only IP addresses. Reverse ARP differs from the Inverse Address Resolution Protocol which is designed to obtain the IP address associated with a local Frame Relay data link connection identifier. InARP is not used in Ethernet

### **ALGORITHM:**

#### **Client**

1. Start the program
2. Create socket and establish connection with the server.
3. Get the IP address to be converted into MAC address from the user.
4. Send this IP address to server.
5. Receive the MAC address for the IP address from the server.
6. Display the received MAC address
7. Terminate the connection

#### **Server**

1. Start the program
2. Create the socket, bind the socket created with IP address and port number and make it a listening socket.
3. Accept the connection request when it is requested by the client.
4. Server maintains the table in which IP and corresponding MAC addresses are stored.
5. Receive the IP address sent by the client.
6. Retrieve the corresponding MAC address for the IP address and send it to the client.
7. Close the connection with the client and now the server becomes a listening server waiting for the connection request from other clients
8. Stop

```
.  
  
import java.io.*;  
import java.net.*;  
import java.util.*;  
class Clientarp  
{  
public static void main(String args[])  
{  
try  
{  
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));  
Socket clsct=new Socket("127.0.0.1",5604)  
DataInputStream din=new DataInputStream(clsct.getInputStream());  
DataOutputStreamdout=new DataOutputStream(clsct.getOutputStream());  
System.out.println("Enter the Logical address(IP):");  
String str1=in.readLine();  
dout.writeBytes(str1+"\n");  
String str=din.readLine();  
  
System.out.println("The Physical Address is: "+str);  
  
clsct.close();  
}  
catch (Exception e)  
{  
System.out.println(e);  
}  
}  
}
```

**Server:**

```
import java.io.*;
import java.net.*;
import java.util.*;

class Serverarp
{
    public static void main(String args[])
    {
        try{
            ServerSocketobj=new
            ServerSocket(139);
            Socket obj1=obj.accept();
            while(true)
            {
                DataInputStream din=new DataInputStream(obj1.getInputStream());
                DataOutputStreamdout=new DataOutputStream(obj1.getOutputStream());
                String str=din.readLine();

                String ip[]={ "165.165.80.80","165.165.79.1"};
                String mac[]={ "6A:08:AA:C2","8A:BC:E3:FA"};
                for(int i=0;i<ip.length;i++)
                {
                    if(str.equals(ip[i]))
                    {
                        dout.writeBytes(mac[i]+'\\n');
                        break;
                    }
                }
                obj.close();
            }
            catch(Exception e)

            {

                System.out.println();
            }
        }
    }
}
```



**Output:**

E:\networks>java Serverarp

E:\networks>java Clientarp

Enter the Logical address(IP):

165.165.80.80

The Physical Address is: 6A:08:AA:C2

**(b) Program for Reverse Address Resolution Protocol (RARP) using UDP**

**ALGORITHM:**

**Client**

1. Start the program
2. Create datagram socket
3. Get the MAC address to be converted into IP address from the user.
4. Send this MAC address to server using UDP datagram.
5. Receive the datagram from the server and display the corresponding IP address.
6. Stop

**Server**

1. Start the program.
2. Server maintains the table in which IP and corresponding MAC addresses are stored.
3. Create the datagram socket
4. Receive the datagram sent by the client and read the MAC address sent.
5. Retrieve the IP address for the received MAC address from the table.
6. Display the corresponding IP address.
7. Stop

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp
{
public static void main(String args[])
{
try
{
DatagramSocket client=new DatagramSocket();
InetAddress addr=InetAddress.getByName("127.0.0.1");
byte[] sendbyte=new byte[1024];
byte[] receivebyte=new byte[1024];
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the Physical address (MAC):");
String str=in.readLine(); sendbyte=str.getBytes();
DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,1309);
client.send(sender);
DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);
client.receive(receiver);
String s=new String(receiver.getData());

System.out.println("The Logical Address is(IP): "+s.trim());
client.close();
```

```
    }  
    catch(Exception e)  
    {  
        System.out.println(e);  
    }  
}
```

**Server:**

```
import java.io.*;  
import java.net.*;  
import java.util.*;  
class Serverarp  
{  
    public static void main(String args[])  
    {  
        try  
        {  
            DatagramSocket server=new DatagramSocket(1309);  
            while(true)  
            {  
                byte[] sendbyte=new byte[1024];  
                byte[] receivebyte=new byte[1024];  
                DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);  
                server.receive(receiver);  
                String str=new String(receiver.getData());  
                String s=str.trim();  
                InetAddressaddr=receiver.getAddress();  
                int port=receiver.getPort();  
                String ip[]={ "165.165.80.80","165.165.79.1"};  
                String mac[]={ "6A:08:AA:C2","8A:BC:E3:FA"};  
                for(int i=0;i<ip.length;i++)  
                {  
                    if(s.equals(mac[i]))  
                    {  
                        sendbyte=ip[i].getBytes();  
                        DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,port);  
                        server.send(sender);  
                        break;  
                    }  
                }  
                break;  
            }  
        }  
        catch(Exception e)  
        {  
            System.out.println(e);  
        }  
    }  
}
```

**Output:**

I:\ex>java Serverrarp12

I:\ex>java Clientrarp12

Enter the Physical address (MAC):

6A:08:AA:C2

The Logical Address is(IP): 165.165.80.80

**7. Implementation of Stop and Wait Protocol and Sliding Window Protocol.**

**Aim: To implement Stop and Wait Protocol and Sliding Window Protocol**

Program:

**Sender:**

```
import java.io.*;

import java.net.*;

import java.util.Scanner;

public class sandwsender {

    public static void main(String args[])

    {

        int p=9040,i,q=8070;

        String h="localhost";

        try

        {

            Scanner scanner = new Scanner(System.in);

            System.out.print("Enter number of frames : ");

            int number = scanner.nextInt();

            if(number==0)

            {

                System.out.println("No frame is sent");

            }

        }

    }

}
```

```
else
{
    Socket s2;

    s2= new Socket(h,q);

    DataOutputStream d1 = new DataOutputStream(s2.getOutputStream());

    d1.write(number);

}

String str1;

for (i=0;i<number;i++)
{
    System.out.print("Enter message : ");

    String name = scanner.next();

    System.out.println("Frame " + i+ " is sent");

    Socket s1;

    s1= new Socket(h,p+i);

    DataOutputStream d = new DataOutputStream(s1.getOutputStream());

    d.writeUTF(name);

    DataInputStream dd= new DataInputStream(s1.getInputStream());

    Integer sss1 = dd.read();

    System.out.println("Ack for : " + sss1 + " is received");

}

}

catch(Exception ex)

{

    System.out.println("ERROR : "+ex);
```

```
    }  
}  
}  
Receiver:  
  
import java.io.*;  
  
import java.net.*;  
  
import java.util.*;  
  
public class sandwreceiver {  
    public static void main(String args[])  
{  
    String h="Serverhost";  
  
    int q=9000;  
  
    int i;  
  
    try  
    {  
ServerSocket ss2;  
  
        ss2 = new ServerSocket(8070);  
  
        Socket s1 =ss2.accept();  
  
        DataInputStream dd1= new DataInputStream(s1.getInputStream());  
  
        Integer i1 =dd1.read();  
  
        for(i=0;i<i1;i++)  
        {  
ServerSocket ss1;  
  
            ss1 = new ServerSocket(9040+i);
```

```
        Socket s = ss1.accept();

DataInputStream dd= new DataInputStream(s.getInputStream());

        String sss1 = dd.readUTF();

System.out.println(sss1);

System.out.println("Frame "+ i+" received");

DataOutputStream d1 = new DataOutputStream(s.getOutputStream());

        d1.write(i);

System.out.println("ACK sent for "+ i);

        }

        }

        catch(Exception ex)

        {

System.out.println("Error"+ex);

        }

    }

}
```

#### OUTPUT:

##### Sender Side

```
Enter number of frames : 3
Enter message : hai
Frame 0 is sent
Ack for :0 is received
Enter message : hello
Frame 1 is sent
Ack for :1 is received
Enter message : how r u?
Frame 2 is sent
Ack for :2 is received
```

Receiver Side:

hai  
Frame 0 received  
ACK sent for 0  
hello  
Frame 1 received  
ACK sent for 1  
how  
Frame 2 received  
ACK sent for 2

### Sliding Window Protocol:

Sender Side:

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Slide_Sender
{
    Socket sender;

    ObjectOutputStream out;
    ObjectInputStream in;

    String pkt;
    char data='a';

    int SeqNum = 1, SWS = 5;
    int LAR = 0, LFS = 0;
    int NF;

    Slide_Sender()
    {
    }

    public void SendFrames()
    {
        if((SeqNum <= 15) && (SWS > (LFS - LAR)))
        {
            try
            {
                NF = SWS - (LFS - LAR);
                for(int i=0; i<NF; i++)
                {
                    pkt = String.valueOf(SeqNum);
```

```
pkt = pkt.concat(" ");
pkt = pkt.concat(String.valueOf(data));
out.writeObject(pkt);
    LFS = SeqNum;
System.out.println("Sent " + SeqNum + " " + data);

    data++;
if(data=='f')
    data='a';

SeqNum++;
out.flush();
    }
    }
catch(Exception e)
    {}
    }
}

public void run() throws IOException
{
    sender = new Socket("localhost",1500);

    out = new ObjectOutputStream(sender.getOutputStream());
    in = new ObjectInputStream(sender.getInputStream());

    while(LAR<15)
    {
        try
        {
            SendFrames();

            String Ack = (String)in.readObject();
            LAR = Integer.parseInt(Ack);
            System.out.println("ack received : " + LAR);
        }
        catch(Exception e)
        {
        }
    }

    in.close();
    out.close();
    sender.close();
    System.out.println("\nConnection Terminated.");
}

public static void main(String as[]) throws IOException
{
    Slide_Sender s = new Slide_Sender();
```



```
s.run();  
    }  
}
```

Receiver Side:

```
import java.io.*;  
import java.net.*;  
import java.util.*;
```

```
public class Slide_Receiver  
{  
    ServerSocket reciever;  
    Socket conc = null;
```

```
    ObjectOutputStream out;  
    ObjectInputStream in;
```

```
    String ack, pkt, data="";  
int delay ;
```

```
int SeqNum = 0, RWS = 5;  
int LFR = 0;  
int LAF = LFR+RWS;
```

```
    Random rand = new Random();
```

```
Slide_Receiver()  
{  
}
```

```
public void run() throws IOException, InterruptedException  
{  
    reciever = new ServerSocket(1500,10);  
    conc = reciever.accept();
```

```
if(conc!=null)  
    System.out.println("Connection established :");
```

```
        out = new ObjectOutputStream(conc.getOutputStream());  
        in = new ObjectInputStream(conc.getInputStream());
```

```
while(LFR<15)  
{
```

```
    try
```

```
        {  
        pkt = (String)in.readObject();  
        String []str = pkt.split("\\s");
```

```
        ack = str[0];
```

```
        data = str[1];

        LFR = Integer.parseInt(ack);

        if((SeqNum<=LFR)||((SeqNum>LAF))
            {
            System.out.println("\nMsg received : "+data);
            delay = rand.nextInt(5);

            if(delay<3 || LFR==15)
                {
                out.writeObject(ack);
                out.flush();
                System.out.println("sending ack " +ack);
                SeqNum++;
                }

            else
                System.out.println("Not sending ack");
                }

            else
                {
                out.writeObject(LFR);
                out.flush();
                System.out.println("resending ack " +LFR);
                }
                }
            }
        catch(Exception e)
            {
            }
        }
    in.close();
    out.close();
    reciever.close();
    System.out.println("\nConnection Terminated.");
    }
    public static void main(String args[]) throws IOException, InterruptedException
    {
    Slide_Receiver R = new Slide_Receiver();
    R.run();
    }
}
```

Output:

Sender Side:

Sent 1 a  
Sent 2 b

Sent 3 c  
Sent 4 d  
Sent 5 e  
ack received : 1  
Sent 6 a  
ack received : 2  
Sent 7 b  
ack received : 3  
Sent 8 c  
ack received : 5  
Sent 9 d  
Sent 10 e  
ack received : 6  
Sent 11 a  
ack received : 7  
Sent 12 b  
ack received : 8  
Sent 13 c  
ack received : 10  
Sent 14 d  
Sent 15 e  
ack received : 12  
ack received : 13  
ack received : 15

Connection Terminated.

Receiver Side:

Msg received : c  
sending ack 3

Msg received : d  
Not sending ack

Msg received : e  
sending ack 5

Msg received : a  
sending ack 6

Msg received : b  
sending ack 7

Msg received : c  
sending ack 8

Msg received : d  
Not sending ack

Msg received : e  
sending ack 10

Msg received : a  
Not sending ack

Msg received : b  
sending ack 12

Msg received : c  
sending ack 13

Msg received : d  
Not sending ack

Msg received : e  
sending ack 15

Connection Terminated.

#### 8. Write a program for congestion control using leaky bucket algorithm.

**Aim: To write a program for congestion control using leaky bucket algorithm.**

```
import java.util.Scanner;
import java.lang.*;
public class LEAKY_BUCKET {
    public static void main(String[] args)
    {
        int i;
        int a[] = new int[20];
        int buck_rem = 0, buck_cap = 4, rate = 3, sent, recv;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of packets");
        int n = in.nextInt();
        System.out.println("Enter the packets");
        for(i = 1; i <= n; i++)
            a[i] = in.nextInt();
        System.out.println("Clock \t packet size \t accept \t sent \t remaining");
        for(i = 1; i <= n; i++)
        {
            if(a[i] != 0)
            {
                if(buck_rem + a[i] > buck_cap)
                    recv = -1;
                else
                {
```

```
recv=a[i];
buck_rem+=a[i];
}
}
else
recv=0;
if(buck_rem!=0)
{
if(buck_rem<rate)
{ sent=buck_rem;
buck_rem=0;
}
else
{
sent=rate;
buck_rem=buck_rem-rate;
}
}
else
sent=0;
if(recv==1)
System.out.println(+i+ "\t\t" +a[i]+ "\t dropped \t" + sent +"\t" +buck_rem);
else
System.out.println(+i+ "\t\t" +a[i] +"\t\t" +recv +"\t" +sent + "\t" +buck_rem);
}
}
}
```

**Experiment No 09.** Implement three node point-to-point networks with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.

TCL:

#Create a simulator object

set ns [ new Simulator ]

#Open the nam trace file

set tf [ open lab1.tr w ]

\$ns trace-all \$tf

#Open the nam trace file

```
set nf [ open lab1.nam w ]
```

```
$ns namtrace-all $nf
```

```
#Define a 'finish' procedure
```

```
proc finish { } { global ns ntf
```

```
$ns flush-trace
```

```
exec nam lab1.nam & close $tf
```

```
close $nf exit 0
```

```
}
```

```
#Creating nodes
```

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

```
Define different colors and labels for data flows
```

```
$ns color 1 "red"
```

```
$ns color 2 "blue"
```

```
$n0 label "Source/udp0"
```

```
$n1 label "Source/udp1"
```

```
$n2 label "Router"
```

```
$n3 label "Destination/Null"
```

```
#Create link between nodes
```

```
$ns duplex-link $n0 $n2 100Mb 300ms DropTail
```

```
$ns duplex-link $n1 $n2 100Mb 300ms DropTail
```

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

```
#Set queue size of links
```

```
$ns set queue-limit $n0 $n2 50
```

```
$ns set queue-limit $n1 $n2 50
```

```
$ns set queue-limit $n2 $n3 5
```

```
#Setup a UDP connection
```

```
set udp0 [new Agent/UDP]
```

```
$ns attach-agent $n0 $udp0
```

```
# Create a CBR traffic source and attach it to udp0
```

```
set cbr0 [new Application/Traffic/CBR]
```

```
$cbr0 set packetSize_ 500
```

```
$cbr0 set interval_ 0.005
```

```
$cbr0 attach-agent $udp0
```

```
#Create a UDP agent and attach it to node n1
```

```
set udp1 [new Agent/UDP]
```

```
$udp1 set class_ 2
```

```
$ns attach-agent $n1 $udp1
```

```
# Create a CBR traffic source and attach it to udp1
```

```
set cbr1 [new Application/Traffic/CBR]
```

```
$cbr1 set packetSize_ 500
```

```
$cbr1 set interval_ 0.005
```

```
$cbr1 attach-agent $udp1
```

```
#Create a Null agent (a traffic sink) and attach it to node n3
```

```
set null0 [new Agent/Null]
```

```
$ns attach-agent $n3 $null0
```

```
#Connect the traffic sources with the traffic sink
```

```
$ns connect $udp0 $null0
```

```
$ns connect $udp1 $null0
```

```
#Schedule events for the CBR agents
```

```
$ns at 0.5 "$cbr0 start"
```

```
$ns at 1.0 "$cbr1 start"
```

```
$ns at 4.0 "$cbr1 stop"
```

```
$ns at 4.5 "$cbr0 stop"
```

```
#Call the finish procedure after 5 seconds of simulation time
```

```
$ns at 5.0 "finish"
```

```
#Run the simulation
```

```
$ns run
```

```
AWK:
```

```
BEGIN{
```

```
count=0;
```

```
}
```

```
{
```

```
if($1=="d") count++
```

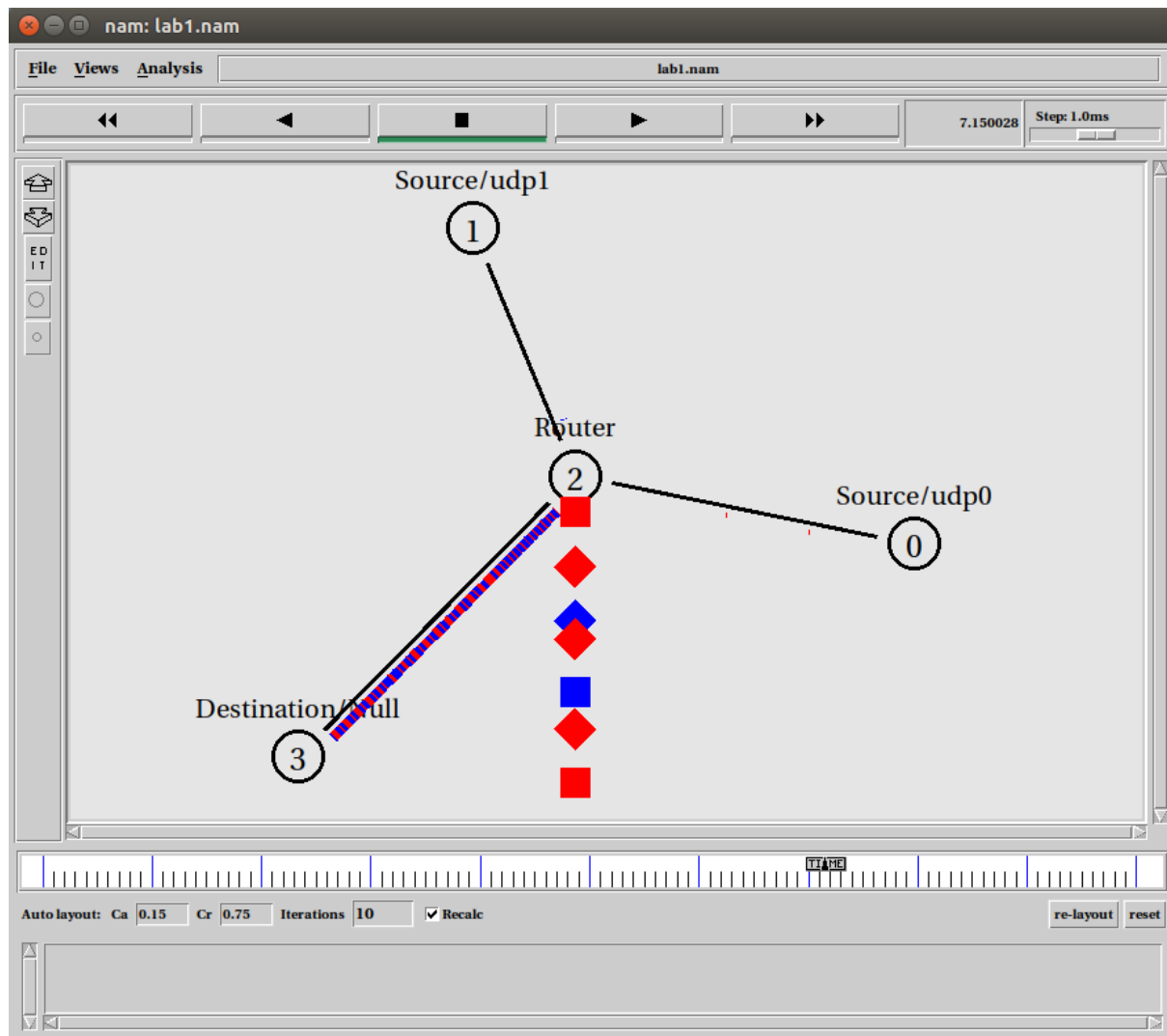
```
} END{
```

```
printf("The Total no of Packets Drop is :%d\n\n", count)
```

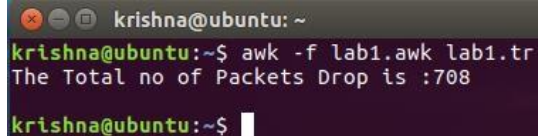
```
}
```



Simulated Network:



Packet Dropped:



A terminal window with a dark background. The prompt is 'krishna@ubuntu: ~'. The command entered is 'awk -f lab1.awk lab1.tr'. The output is 'The Total no of Packets Drop is :708'. The prompt is then 'krishna@ubuntu:~\$' followed by a cursor.

```
krishna@ubuntu: ~  
krishna@ubuntu:~$ awk -f lab1.awk lab1.tr  
The Total no of Packets Drop is :708  
krishna@ubuntu:~$
```

OR

Using grep command: `cat lab1.tr | grep ^d | wc -l`

**10. Simulate the transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion**

**Aim:Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.**

#Create Simulator

set ns [new Simulator]

#Use colors to differentiate the traffic

\$ns color 1 Blue

\$ns color 2 Red

#Open trace and NAM trace file set ntrace [open prog3.tr w]

\$ns trace-all \$ntrace

set namfile [open prog3.nam w]

\$ns namtrace-all \$namfile

```
#Finish Procedure proc Finish { } {  
    global ns ntracenamfile  
  
    #Dump all trace data and close the file  
    $ns flush-trace close $ntrace close $namfile  
  
    #Execute the nam animation file exec nam prog3.nam &  
  
    #Find the number of ping packets dropped  
    puts "The number of ping packets dropped are "  
    exec grep "^d" prog3.tr | cut -d " " -f 5 | grep -c "ping" & exit 0  
}  
  
#Create six nodes  
for {set i 0} {$i < 6} {incr i} {  
    set n($i) [$ns node]  
}  
  
#Connect the nodes  
for {set j 0} {$j < 5} {incr j} {  
    $ns duplex-link $n($j) $n([expr ($j+1)]) 0.1Mb 10ms DropTail  
}  
  
#Define the recv function for the class 'Agent/Ping'  
Agent/Ping instprocrecv {from rtt} {
```

```
$self instvar node_  
puts "node [$node_ id] received ping answer from $from with round trip time $rtt  
ms"  
}
```

```
#Create two ping agents and attach them to n(0) and n(5)
```

```
set p0 [new Agent/Ping]
```

```
$p0 set class_ 1
```

```
$ns attach-agent $n(0) $p0
```

```
set p1 [new Agent/Ping]
```

```
$p1 set class_ 1
```

```
$ns attach-agent $n(5) $p1
```

```
$ns connect $p0 $p1
```

```
#Set queue size and monitor the queue
```

```
#Queue size is set to 2 to observe the drop in ping packets
```

```
$ns queue-limit $n(2) $n(3) 2
```

```
$ns duplex-link-op $n(2) $n(3) queuePos 0.5
```

```
#Create Congestion
```

```
#Generate a Huge CBR traffic between n(2) and n(4)
```

```
set tcp0 [new Agent/TCP]
```

```
$tcp0 set class_ 2
```

```
$ns attach-agent $n(2) $tcp0 set sink0 [new Agent/TCPSink]
```

\$ns attach-agent \$n(4) \$sink0

\$ns connect \$tcp0 \$sink0

#Apply CBR traffic over TCP

set cbr0 [new Application/Traffic/CBR]

\$cbr0 set packetSize\_ 500

\$cbr0 set rate\_ 1Mb

\$cbr0 attach-agent \$tcp0

#Schedule events

\$ns at 0.2 "\$p0 send"

\$ns at 0.4 "\$p1 send"

\$ns at 0.4 "\$cbr0 start"

\$ns at 0.8 "\$p0 send"

\$ns at 1.0 "\$p1 send"

\$ns at 1.2 "\$cbr0 stop"

\$ns at 1.4 "\$p0 send"

\$ns at 1.6 "\$p1 send"

\$ns at 1.8 "Finish"

#Run the Simulation

\$ns run

**11.Simulate an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.**

#Create Simulator

set ns [new Simulator]

#Use colors to differentiate the traffics

\$ns color 1 Blue

\$ns color 2 Red

#Open trace and NAM trace file set ntrace [open prog5.tr w]

\$ns trace-all \$ntrace

set namfile [open prog5.nam w]

\$ns namtrace-all \$namfile

#Use some flat file to create congestion graph windows set winFile0 [open WinFile0 w]

set winFile1 [open WinFile1 w]

#Finish Procedure proc Finish { } {

#Dump all trace data and Close the files global ns ntracenamfile

\$ns flush-trace close \$ntrace close \$namfile

#Execute the NAM animation file exec nam prog5.nam &

```
#Plot the Congestion Window graph using xgraph exec xgraph WinFile0 WinFile1 &
exit 0
}

#Plot Window Procedure

proc PlotWindow {tcpSource file} { global ns

set time 0.1

set now [$ns now]

set cwnd [$tcpSource set cwnd_] puts $file "$now $cwnd"

$ns at [expr $now+$time] "PlotWindow $tcpSource $file"

}

#Create 6 nodes

for {set i 0} {$i<6} {inc i} { set n($i) [$ns node]

}

#Create duplex links between the nodes

$ns duplex-link $n(0) $n(2) 2Mb 10ms DropTail

$ns duplex-link $n(1) $n(2) 2Mb 10ms DropTail

$ns duplex-link $n(2) $n(3) 0.6Mb 100ms DropTail

#Nodes n(3) , n(4) and n(5) are considered in a LAN

set lan [$ns newLan "$n(3) $n(4) $n(5)" 0.5Mb 40ms LL Queue/DropTail MAC/802_3 Channel]
```

#Orientation to the nodes

\$ns duplex-link-op \$n(0) \$n(2) orient right-down

\$ns duplex-link-op \$n(1) \$n(2) orient right-up

\$ns duplex-link-op \$n(2) \$n(3) orient right

#Setup queue between n(2) and n(3) and monitor the queue

\$ns queue-limit \$n(2) \$n(3) 20

\$ns duplex-link-op \$n(2) \$n(3) queuePos 0.5

#Set error model on link n(2) to n(3) set loss\_module [new ErrorModel]

\$loss\_moduleranvar [new RandomVariable/Uniform]

\$loss\_module drop-target [new Agent/Null]

\$ns lossmodel \$loss\_module \$n(2) \$n(3)

#Set up the TCP connection between n(0) and n(4) set tcp0 [new Agent/TCP/Newreno]

\$tcp0 set fid\_ 1

\$tcp0 set window\_ 8000

\$tcp0 set packetSize\_ 552

\$ns attach-agent \$n(0) \$tcp0

set sink0 [new Agent/TCPSink/DelAck]

\$ns attach-agent \$n(4) \$sink0

\$ns connect \$tcp0 \$sink0



```
#Apply FTP Application over TCP set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

$ftp0 set type_ FTP

#Set up another TCP connection between n(5) and n(1) set tcp1 [new Agent/TCP/Newreno]

$tcp1 set fid_ 2

$tcp1 set window_ 8000

$tcp1 set packetSize_ 552

$ns attach-agent $n(5) $tcp1

set sink1 [new Agent/TCPSink/DelAck]

$ns attach-agent $n(1) $sink1

$ns connect $tcp1 $sink1

#Apply FTP application over TCP set ftp1 [new Application/FTP]

$ftp1 attach-agent $tcp1

$ftp1 set type_ FTP

#Schedule Events

$ns at 0.1 "$ftp0 start"

$ns at 0.1 "PlotWindow $tcp0 $winFile0"

$ns at 0.5 "$ftp1 start"

$ns at 0.5 "PlotWindow $tcp1 $winFile1"

$ns at 25.0 "$ftp0 stop"

$ns at 25.1 "$ftp1 stop"

$ns at 25.2 "Finish"
```

#Run the simulation

\$ns run

**Experiment No12.Simulate simple ESS and with transmitting nodes in wireless LAN by simulation and determine the performance with respect to transmission of packets.**

**Aim:Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.**

#Code to be saved as .tcl

#Create a ns simulator

set ns [new Simulator]

#Setup topography object

set topo [new Topography]

\$topoload\_flatgrid 1500 1500

#Open the NS trace file

set tracefile [open p4.tr w]

\$ns trace-all \$tracefile

#Open the NAM trace file

set namfile [open p4.nam w]

\$ns namtrace-all \$namfile

\$ns namtrace-all-wireless \$namfile 1500 1500

#=====

#Mobile node parameter setup

#=====

\$ns node-config -adhocRouting DSDV \

-llType LL \

-macType Mac/802\_11 \

-ifqType Queue/DropTail \

-ifqLen 20 \

-phyTypePhy/WirelessPhy \

-channelType Channel/WirelessChannel \

-propType Propagation/TwoRayGround \

-antType Antenna/OmniAntenna \

-topoInstance \$topo \

-agentTrace ON \

-routerTrace ON

#=====

#Nodes Definition

#=====

create-god 6

#Create 6 nodes

set n0 [\$ns node]

\$n0 set X\_ 630

\$n0 set Y\_ 501

\$n0 set Z\_ 0.0

\$ns initial\_node\_pos \$n0 20

set n1 [\$ns node]

\$n1 set X\_ 454

\$n1 set Y\_ 340

\$n1 set Z\_ 0.0

\$ns initial\_node\_pos \$n1 20

set n2 [\$ns node]

\$n2 set X\_ 785

\$n2 set Y\_ 326

\$n2 set Z\_ 0.0

\$ns initial\_node\_pos \$n2 20

set n3 [\$ns node]

\$n3 set X\_ 270

\$n3 set Y\_ 190

\$n3 set Z\_ 0.0

\$ns initial\_node\_pos \$n3 20

set n4 [\$ns node]

\$n4 set X\_ 539

\$n4 set Y\_ 131

\$n4 set Z\_ 0.0

\$ns initial\_node\_pos \$n4 20

set n5 [\$ns node]

\$n5 set X\_ 964

\$n5 set Y\_ 177

\$n5 set Z\_ 0.0

\$ns initial\_node\_pos \$n5 20

#=====

#Agents Definition

#=====

#Setup a UDP connection

set udp0 [new Agent/UDP]

\$ns attach-agent \$n0 \$udp0

set null1 [new Agent/Null]

\$ns attach-agent \$n4 \$null1

\$ns connect \$udp0 \$null1

\$udp0 set packetSize\_ 1500

#Setup a TCP connection

set tcp0 [new Agent/TCP]

\$ns attach-agent \$n3 \$tcp0

set sink1 [new Agent/TCPSink]

\$ns attach-agent \$n5 \$sink1

\$ns connect \$tcp0 \$sink1

#=====

#Applications Definition

#=====

#Setup a CBR Application over UDP connection

set cbr0 [new Application/Traffic/CBR]

\$cbr0 attach-agent \$udp0

\$cbr0 set packetSize\_ 1000

\$cbr0 set rate\_ 1.0Mb

\$cbr0 set random\_ null

#Setup a FTP Application over TCP connection

set ftp0 [new Application/FTP]

\$ftp0 attach-agent \$tcp0

#=====

#Termination

#=====

#Define a 'finish' procedure

proc finish { } {

global ns tracefile

\$ns flush-trace

close \$tracefile

close \$namfile

exec nam p4.nam &

exec echo "Number of packets dropped is:" &

exec grep -c "^D" p4.tr &

exit 0

}

\$ns at 1.0 "\$cbr0 start"

\$ns at 2.0 "\$ftp0 start"

\$ns at 180.0 "\$ftp0 stop"

\$ns at 200.0 "\$cbr0 stop"

\$ns at 200.0 "finish"

\$ns at 70 "\$n4 set dest 100 60 20"

\$ns at 100 "\$n4 set dest 700 300 20"

\$ns at 150 "\$n4 set dest 900 200 20"

\$ns run

#=====

#CODE TO BE SAVED AS .awk FILE

#=====

BEGIN{

count1=0

count2=0

pack1=0

pack2=0

time1=0

time2=0

}

```
{  
if($1=="r"&&$3=="_1_"&&$4=="RTR")  
{  
count1++  
pack1=pack1+$8  
time1=$2  
}  
if($1=="r"&&$3=="_2_"&&$4=="RTR")  
{  
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: %fMbps\n",((count2*pack2*8)/(time2*1000000)));  
}
```

### VIVA QUESTIONS

- 1.What is socket? How is it implemented?
- 2.What is the different system calls on client side and server side implementation?
- 3.How to convert single client into multiple clients handling in server?
- 4.What are the differences between SOCK\_STREAM, SOCK\_DGRAM, and SOCK\_RAM?  
What is their level of implementation?
- 5.Explain need of sock adder\_ in structure and IN ADDR\_ANY.



- 6.What is FIFO? How to implement Multiple FIFOs. Disadvantages of FIFO?
- 7.Which are the different IPC Primitives? Explain Why IPC Primitives are required?
- 8.Explain the purpose of RS algorithm? What is Public and Private Key? How to find public and private key in RSA algorithm?
- 9.List some Cryptographic algorithms?
- 10.What is congestion? Explain leaky bucket and token bucket algorithm? What are its disadvantages of each?
- 11.List other congestion control algorithm.
- 12.What are the disadvantages of RSA Algorithm?
- 13.What is the need for routing protocols? List some routing protocols. Which are common routing algorithms used by routing protocols
- 14.Disadvantages of Distance vector routing. Explain the method to overcome such disadvantage.
- 15.What is CRC? What is the need for CRC? Which Layer it is implemented?
- 16.Can CRC be employed for error correction if so how.
- 17.Explain steps of error correction and error detection. What is encoding and decoding, why it is required?
- 18.What is Hamming code? Can it be used for error detection and correction how?
- 19.What are linear block codes? Explain their advantages.
- 20.What is protocol? List some protocols. How to modify existing protocol?
- 21.Difference between TCP/IP and OSI Layer Model?
- 22.Difference between Router, Bridge, Gateway channel. In which layer it is used?
- 23.Difference between hub and switch. Which is more compatible and reliable?
- 24.Explain various Ethernet standards.
- 25.Difference between IP and MAC address.
- 26.In which layer ATM protocol is used? Explain the purpose.
- 27.In Which layer or protocol which converts IP to MAC Address.
- 28.Which are the different collision detection protocols?
- 29.Which is the different channelization protocol and explain their needs?
- 30.Explain term error rate, bandwidth, throughput transmission delay, BER, Interarrival time. What is unit of bandwidth?

### **Do's**

Do wear ID card and follow dress code.

- Do log off the computers when you finish.

- Do ask for assistance if you need help.
- Do keep your voice low when speaking to others in the LAB.
- Do ask for assistance in downloading any software.
- Do make suggestions as to how we can improve the LAB.
- In case of any hardware related problem, ask LAB in charge for solution.
- If you are the last one leaving the LAB, make sure that the staff in charge of the LAB is informed to close the LAB.
- Be on time to LAB sessions.
- Do keep the LAB as clean as possible.

### **Don'ts**

- Do not use mobile phone inside the lab.
- Don't do anything that can make the LAB dirty (like eating, throwing waste papers etc).
- Do not carry any external devices without permission.
- Don't move the chairs of the LAB.
- Don't interchange any part of one computer with another.
- Don't leave the computers of the LAB turned on while leaving the LAB.
- Do not install or download any software or modify or delete any system files on any lab computers.
- Do not damage, remove, or disconnect any labels, parts, cables, or equipment.
- Don't attempt to bypass the computer security system.
- Do not read or modify other user's file.
- If you leave the lab, do not leave your personal belongings unattended. We are not responsible for any theft.
- Do not install or download any software or modify or delete any system files on any lab computers.
- Do not damage, remove, or disconnect any labels, parts, cables, or equipment.
- Don't attempt to bypass the computer security system.
- Do not read or modify other user's file.
- If you leave the lab, do not leave your personal belongings unattended. We are not responsible for any theft.