



### Overview

<b>Year / Semester</b>	3 <sup>rd</sup> Year		<b>Academic Year</b>	2016 - 2017
<b>Laboratory Title</b>	Computer Network Laboratory		<b>Laboratory Code</b>	15CSL57
<b>Total Contact Hours</b>	40 Hours		<b>Duration of SEE</b>	3 Hours
<b>IA Marks</b>	20 Marks		<b>SEE Marks</b>	80 Marks
<b>Lab Manual Author</b>	Prof. M. A. Chitale	<b>Sign -</b>	<b>Date</b>	
<b>Checked By</b>		<b>Sign -</b>	<b>Date</b>	

### Objectives

1. To plan the laboratory activities as to complete the experiments on time.
2. To write & complete the records at the end of each experiment.
3. To prepare the laboratory manual.

### Description

#### 1.0 Learning Objectives

- Demonstrate operation of network and its management commands
- Simulate and demonstrate the performance of GSM and CDMA
- Implement data link layer and transport layer protocols.

#### 2.0 Learning Outcomes

- Analyze and Compare various networking protocols.
- Implement, analyze and evaluate networking protocols in NS2 / NS3.
- Demonstrate the working of different concepts of networking.

### Prerequisites

The Lab requires the student to have the following prerequisites:

1. Student should have the basic knowledge of JAVA and TCL scripting.
2. Student should have the basic knowledge of Computer Networks algorithms like error detection and correction, security, congestion control and routing algorithms.
3. Student should have the basic knowledge of networking protocols like TCP/IP, UDP, FTP and TELNET.
4. The student should have knowledge of TCP/IP sockets programming.

### Base Course



1. Networks Laboratory

### Introduction

- A computer network, or simply a network, is a collection of computers and other hardware components interconnected by communication channels that allow sharing of resources and information.
- Where at least one process in one device is able to send/receive data to/from at least one process residing in a remote device, then the two devices are said to be in a network.
- Simply, more than one computer interconnected through a communication medium for information interchange is called a computer network.
- Communications protocols define the rules and data formats for exchanging information in a computer network, and provide the basis for network programming.
- In this course we will study simulation of networking protocols such as TCP, UDP, FTP, TELNET, ETHERNET etc.
- We will study implementation of networking algorithms such as error detection and correction, routing, security etc in 'C'.

### Resources Required

1. The lab should have:
2. Intel based PC
3. gcc compiler
4. NS2 simulation tool
5. Word processor

### General Instructions

1. Student should be punctual to the Lab
2. Required to prepare the Lab report every week
3. Required to maintain the Lab record properly
4. Should use the resources properly
5. Required to prepare the Lab report every week
6. Required to maintain the Lab record properly
7. Should use the resources properly

### Contents

Expt No.	Experiments	Date Planned	Date Conducted
PART-A			
1	Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.		
2	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.		
3	Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.		
4	Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.		
5	Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment		
6	Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment.		
<b>PART-B</b>			
7	Write a program for error detecting code using CRC-CCITT (16- bits).		
8	Write a program to find the shortest path between vertices using bellman-ford algorithm		
9	Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.		
10	Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.		
11	Write a program for simple RSA algorithm to encrypt and decrypt the data.		
12	Write a program for congestion control using leaky bucket algorithm.		

#### **EvaluationScheme**

1. Lab activity – 1 Mark allotted to each experiment – 15 Marks.
2. Internal exam at the end of sem – 10 Marks.
3. Sem End Exam – 50 Marks.

#### **Reference**



1. Communication Networks – Fundamental Concepts and Key architectures – Alberto Leon-Garcia and Indra Widjaja, 2<sup>nd</sup> Edition, Tata McGraw-Hill, 2004.
2. Computer Networks A Systems Approach – Larry L. Peterson and Bruce S. David – 4<sup>th</sup> Edition, Elsevier, 2007.
3. Data and Computer Communication, William Stallings, 8<sup>th</sup> Edition, Pearson Education, 2007.

## Experiments

### 1.0 Experiment

Experiment No	Date Planed	Date Conducted	Marks
01			

#### **TITLE:**

Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.

#### 1.1 Learning Objectives

- To make student comfortable in using NS2 tool.
- Simulate point to point network.

#### 1.2 Aim

Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.

#### 1.3 Material / Equipment Required

#### 1.4 Theory / Hypothesis

The Point-to-Point Protocol (PPP) originally emerged as an encapsulation protocol for transporting IP traffic over point-to-point links. PPP also established a standard for the assignment and management of IP addresses, asynchronous (start/stop) and bit-oriented synchronous encapsulation, network protocol multiplexing, link configuration, link quality testing, error detection, and option negotiation for such capabilities as network layer address negotiation and data-compression negotiation. PPP supports these functions by providing an extensible Link Control Protocol (LCP) and a family of Network Control Protocols (NCPs) to negotiate optional configuration parameters and facilities. In addition to IP, PPP supports other protocols, including Novell's Internetwork Packet Exchange (IPX) and DECnet. PPP provides a method for transmitting datagrams over serial point-to-point links. PPP contains three main components:

1. A method for encapsulating datagrams over serial links. PPP uses the High-Level Data Link Control (HDLC) protocol as a basis for encapsulating datagrams over point-to-point links.
2. An extensible LCP to establish, configure, and test the data link connection.



3. A family of NCPs for establishing and configuring different network layer protocols. PPP is designed to allow the simultaneous use of multiple network layer protocols. To establish communications over a point-to-point link, the originating PPP first sends LCP frames to configure and (optionally) test the data link. After the link has been established and optional facilities have been negotiated as needed by the LCP, the originating PPP sends NCP frames to choose and configure one or more network layer protocols. When each of the chosen network layer protocols has been configured, packets from each network layer protocol can be sent over the link. The link will remain configured for communications until explicit LCP or NCP frames close the link, or until some external event occurs (for example, an inactivity timer expires or a user intervenes).

### **1.5 Procedure / Program / Activity**

Program:

```
set ns [new Simulator]
set nf [open expt1.nam w]
$ns namtrace-all $nf

set nd [open expt1.tr w]
$ns trace-all $nd

proc finish { } {
    global ns nf nd
    $ns flush-trace
    close $nf
    close $nd
    exec nam expt1.nam &
    exit 0
}

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

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 512Kb 10ms DropTail
$ns queue-limit $n0 $n2 5
$ns queue-limit $n1 $n2 5

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```



```
set sink [new Agent/Null]
$ns attach-agent $n1 $sink
$ns connect $udp0 $sink
$ns at 0.2 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

AWK file

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

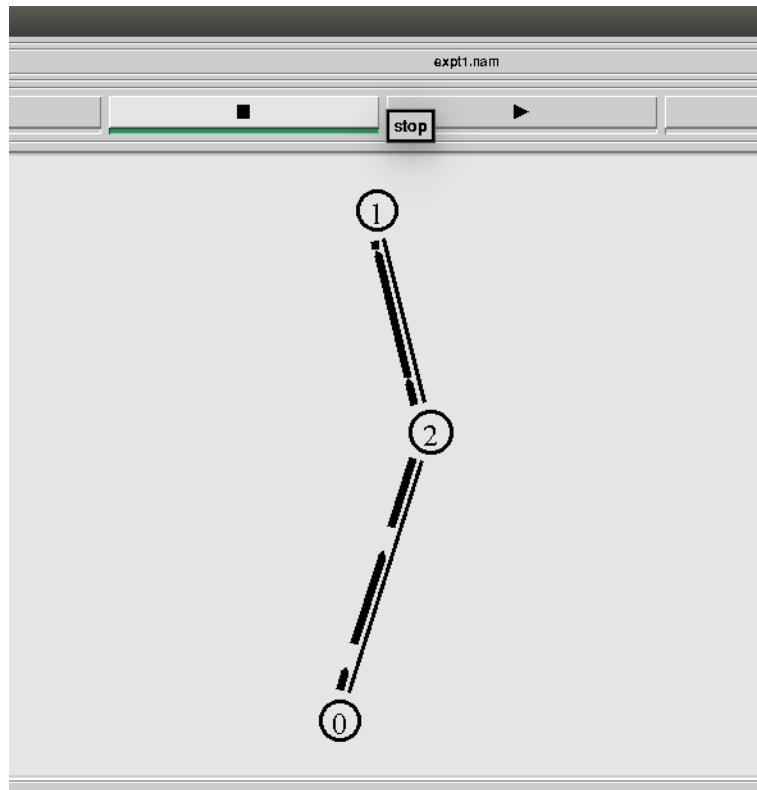
Procedure:

- 1) Open vi editor and type program. Program name should have the extension “**.tcl**”  
**[root@localhost ~]# vi lab1.tcl**
- 2) Save the program by pressing “**ESC key**” first, followed by “**Shift and :**” keys simultaneously and type “**wq**” and press **Enterkey**.
- 3) Open vi editor and type **awk** program. Program name should have the extension “**.awk**”  
**[root@localhost ~]# vi lab1.awk**
- 4) Save the program by pressing “**ESC key**” first, followed by “**Shift and :**” keys simultaneously and type “**wq**” and press **Enterkey**.
- 5) Run the simulation program  
**[root@localhost~]# ns lab1.tcl**
  - i) Here “**ns**” indicates network simulator. We get the topology shown in the snapshot.
  - ii) Now press the play button in the simulation window and the simulation will begin.
- 6) After simulation is completed run **awk file** to see the output,  
**[root@localhost~]# awk -f lab1.awk lab1.tr**
- 7) To see the trace file contents open the file as,

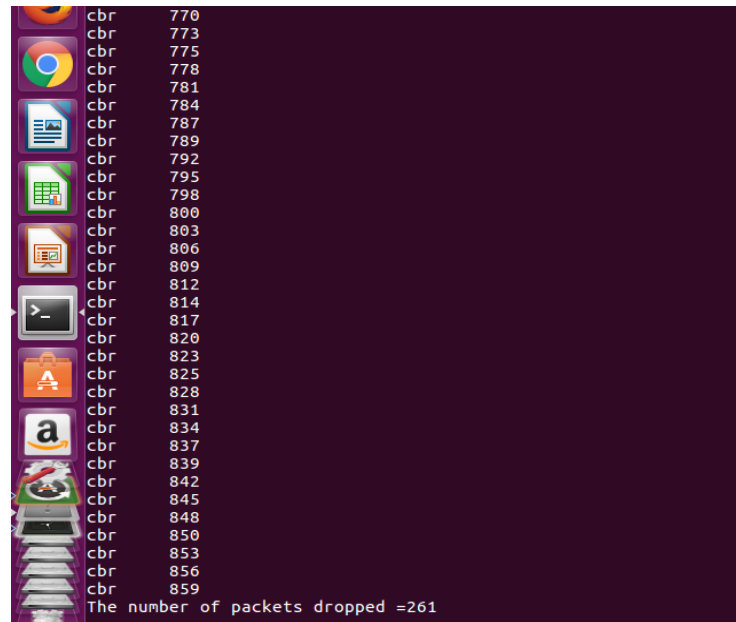


```
[root@localhost~]# vi lab1.tr
```

## 1.6 Block / Circuit / Model / Reaction Diagram



## 1.7 Results & Analysis



```
cbr 770
cbr 773
cbr 775
cbr 778
cbr 781
cbr 784
cbr 787
cbr 789
cbr 792
cbr 795
cbr 798
cbr 800
cbr 803
cbr 806
cbr 809
cbr 812
cbr 814
cbr 817
cbr 820
cbr 823
cbr 825
cbr 828
cbr 831
cbr 834
cbr 837
cbr 839
cbr 842
cbr 845
cbr 848
cbr 850
cbr 853
cbr 856
cbr 859
The number of packets dropped =261
```

## 1.8 Outcome & Conclusion

- As the source starts sending packets, over period time the buffer becomes full and starts discarding packets.

## 1.9 Remarks

- Number of packet drop occurs due to buffer overflow.

Faculty Signature





## **1.0 Experiment**

Experiment No	Date Planed	Date Conducted	Marks
02			

### **TITLE:**

**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.**

## **1.1 Learning Objectives**

- To understand transmission of ping messages.

## **1.2 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.

## **1.3 Material / Equipment Required**



## **1.4 Theory / Hypothesis**

Most network administrators are very familiar with the ping utility and are likely to use it on an almost daily basis. The basic function of the ping command is to test the connectivity between the two devices on a network. All the command is designed to do is determine whether the two computers can see each other and to notify you of how long the round-trip takes to complete.

Although ping is most often used on its own, a number of switches can be used to assist in the troubleshooting process. Table 3 shows some of the commonly used switches with ping on a Windows system. Ping works by sending ICMP echo request messages to another device on the network. If the other device on the network hears the ping request, it automatically responds with an ICMP echo reply. By default, the ping command on a Windows-based system sends four data packets; however, using the -t - switch, a continuous stream of pingrequests can be sent. pingis perhaps the most widely used of all network tools; it is primarily used to verify connectivity between two network devices. On a good day, the results from the ping command will be successful, and the sending device will receive a reply from the remote device. Not all ping results are that successful, and to be able to effectively use ping, you must be able to interpret the results of a failed ping command.

## **1.5 Procedure / Program / Activity**

### **Program:**

```
set ns [new Simulator]
set nf [open expt2.nam w]
$ns namtrace-all $nf
set nd [open expt2.tr w]
$ns trace-all $nd
```

```
proc finish { } {
    global ns nf nd
    $ns flush-trace
    close $nf
    close $nd
    exec nam expt2.nam &
    exit 0
}
```

```
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]
```

```
$ns duplex-link $n1 $n0 1Mb 1ms DropTail
```



```
$ns duplex-link $n2 $n0 1Mb 1ms DropTail
$ns duplex-link $n3 $n0 1Mb 1ms DropTail
$ns duplex-link $n4 $n0 1Mb 1ms DropTail
$ns duplex-link $n5 $n0 1Mb 1ms DropTail
$ns duplex-link $n6 $n0 256Kb 200ms DropTail
```

```
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] recieved ping answer from $from with round-trip-time $rtt ms."
}
```

```
set p1 [new Agent/Ping]
set p2 [new Agent/Ping]
set p3 [new Agent/Ping]
set p4 [new Agent/Ping]
set p5 [new Agent/Ping]
set p6 [new Agent/Ping]
set p7 [new Agent/Ping]
set p8 [new Agent/Ping]
set p9 [new Agent/Ping]
```

```
$ns attach-agent $n1 $p1
$ns attach-agent $n2 $p2
$ns attach-agent $n3 $p3
$ns attach-agent $n4 $p4
$ns attach-agent $n5 $p5
$ns attach-agent $n6 $p6
$ns attach-agent $n6 $p7
$ns attach-agent $n6 $p8
$ns attach-agent $n6 $p9
```

```
$ns queue-limit $n0 $n1 3
$ns queue-limit $n0 $n2 3
$ns queue-limit $n0 $n3 3
$ns queue-limit $n0 $n4 2
$ns queue-limit $n0 $n5 2
$ns queue-limit $n0 $n6 2
```

```
$ns connect $p1 $p6
$ns connect $p2 $p6
$ns connect $p3 $p7
$ns connect $p4 $p8
```

```
$ns at 0.1 "$p1 send"
$ns at 0.11 "$p2 send"
$ns at 0.11 "$p3 send"
```



\$ns at 0.11 "\$p4 send"

\$ns at 0.2 "\$p1 send"

\$ns at 0.2 "\$p2 send"

\$ns at 0.3 "\$p3 send"

\$ns at 0.4 "\$p4 send"

\$ns at 0.6 "\$p1 send"

\$ns at 0.7 "\$p2 send"

\$ns at 0.7 "\$p3 send"

\$ns at 0.7 "\$p4 send"

\$ns at 0.10 "\$p1 send"

\$ns at 0.11 "\$p2 send"

\$ns at 0.12 "\$p3 send"

\$ns at 0.11 "\$p4 send"

\$ns at 0.13 "\$p1 send"

\$ns at 0.13 "\$p2 send"

\$ns at 0.14 "\$p3 send"

\$ns at 0.14 "\$p4 send"

\$ns at 0.15 "\$p1 send"

\$ns at 0.15 "\$p2 send"

\$ns at 0.16 "\$p3 send"

\$ns at 0.16 "\$p4 send"

\$ns at 3.0 "finish"

\$ns run

**AWK File:**

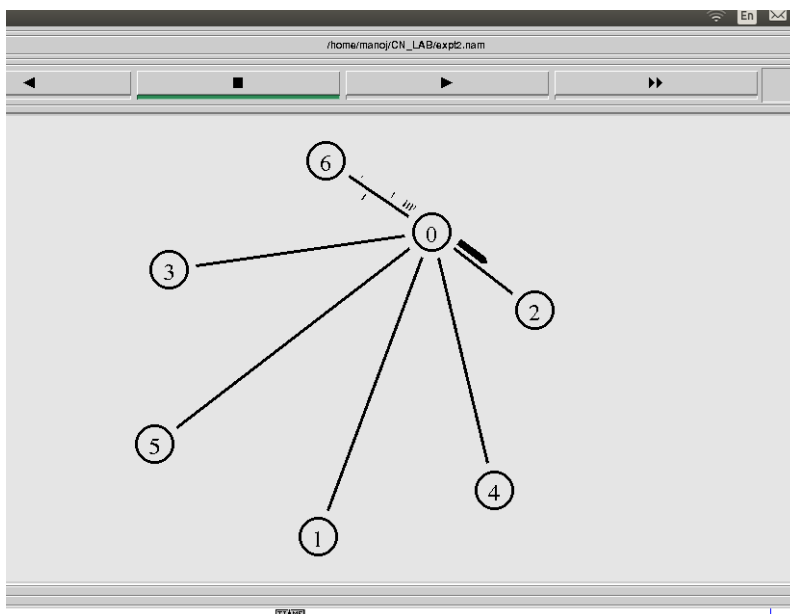
```
BEGIN {  
    c=0;  
}  
{  
    if($1=="d")  
    {  
        c++;  
        printf("%s\t%s\n",$5,$11);  
    }  
}  
END{  
    printf("The number of packets dropped =%d\n",c);  
}
```

Procedure:



- 1) Open vi editor and type program. Program name should have the extension “.tcl”  
**[root@localhost ~]# vi lab2.tcl**
- 2) Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press **Enterkey**.
- 3) Open vi editor and type **awk** program. Program name should have the extension “.awk”  
**[root@localhost ~]# vi lab2.awk**
- 4) Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press **Enterkey**.
- 5) Run the simulation program  
**[root@localhost~]# ns lab2.tcl**
  - i) Here “ns” indicates network simulator. We get the topology shown in the snapshot.
  - ii) Now press the play button in the simulation window and the simulation will begin.
- 6) After simulation is completed run **awk file** to see the output,  
**[root@localhost~]# awk -f lab2.awk lab2.tr**
- 7) To see the trace file contents open the file as,  
**[root@localhost~]# vi lab2.tr**

## 1.6 Block / Circuit / Model / Reaction Diagram



## 1.7 Results & Analysis



```
manoj@manoj-Inspiron-N5110: ~/CN_LAB
manoj@manoj-Inspiron-N5110:~/CN_LAB$ awk -f expt1.awk expt2.tr
The number of packets dropped =4
manoj@manoj-Inspiron-N5110:~/CN_LAB$
```

## 1.8 Outcome & Conclusion

- As the source nodes starts sending ping messages continuously the intermediate packet switch becomes congested and due to bottleneck effect the switch starts discarding the packets.

## 1.9 Remarks

- Packet drop occurs due to congestion.

**FACULTY SIGNATURE**



## 1.0 Experiment

Experiment No	Date Planed	Date Conducted	Marks
03			

### TITLE:

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

## 1.1 Learning Objectives

- To understand TCP congestion control mechanism.

## 1.2 Aim

- Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source /destination.

## 1.3 Material / Equipment Required

## 1.4 Theory / Hypothesis

The TCP congestion control mechanism has each side of the connection keep track of two additional variables: the **congestion window** and the **threshold**. The congestion window, denoted *CongWin*, imposes an additional constraint on how much traffic a host can send into a connection. Specifically, the amount of unacknowledged data that a host can have within a TCP connection may not exceed the minimum of *CongWin* and *RcvWin*, i.e.,

$LastByteSent - LastByteAcked \leq \min\{CongWin, RcvWin\}$ .

Once a TCP connection is established between the two end systems, the application process at the sender writes bytes to the sender's TCP send buffer. TCP grabs chunks of size MSS, encapsulates each chunk within a TCP segment, and passes the segments to the network layer for transmission across the network. The TCP congestion window regulates the times at which the segments are sent into the network (i.e., passed to the network layer). Initially, the congestion window is equal to one MSS. TCP sends the first segment into the network and waits for an acknowledgement. If this segment is acknowledged before its timer times out, the sender increases the congestion window by one MSS and sends out two maximum-size segments. If these segments are acknowledged before their timeouts, the sender increases the congestion window by one MSS for each of the acknowledged segments, giving a congestion window of four MSS, and sends out four maximum-sized segments. This procedure continues as long as (1) the congestion window is below the threshold and (2) the acknowledgements arrive before their corresponding timeouts.

## 1.5 Procedure / Program / Activity



**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]
$n0 color "magenta"
$n0 label "src1"

set n1 [$ns node]
set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2"
set n3 [$ns node]
$n3 color "blue"
$n3 label "dest2"
set n4 [$ns node]
set n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail Mac/802_3
$ns duplex-link $n4 $n5 1Mb 1ms DropTail
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001
set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5
$ns connect $tcp0 $sink5
set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001

set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp2 $sink3

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





```
$tcp0 attach $file1
```

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

```
$tcp2 attach $file2
```

```
$tcp0 trace cwnd_
```

```
$tcp2 trace cwnd_
```

```
proc finish { } {
```

```
global ns nf tf
```

```
$ns flush-trace
```

```
close $tf
```

```
close $nf
```

```
exec nam lab3.nam &
```

```
exit 0
```

```
}
```

```
$ns at 0.1 "$ftp0 start"
```

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

```
$ns at 7 "$ftp0 start"
```

```
$ns at 0.2 "$ftp2 start"
```

```
$ns at 8 "$ftp2 stop"
```

```
$ns at 14 "$ftp0 stop"
```

```
$ns at 10 "$ftp2 start"
```

```
$ns at 15 "$ftp2 stop"
```

```
$ns at 16 "finish"
```

```
$ns run
```

AWK File:

```
BEGIN {
```

```
}
```

```
{
```

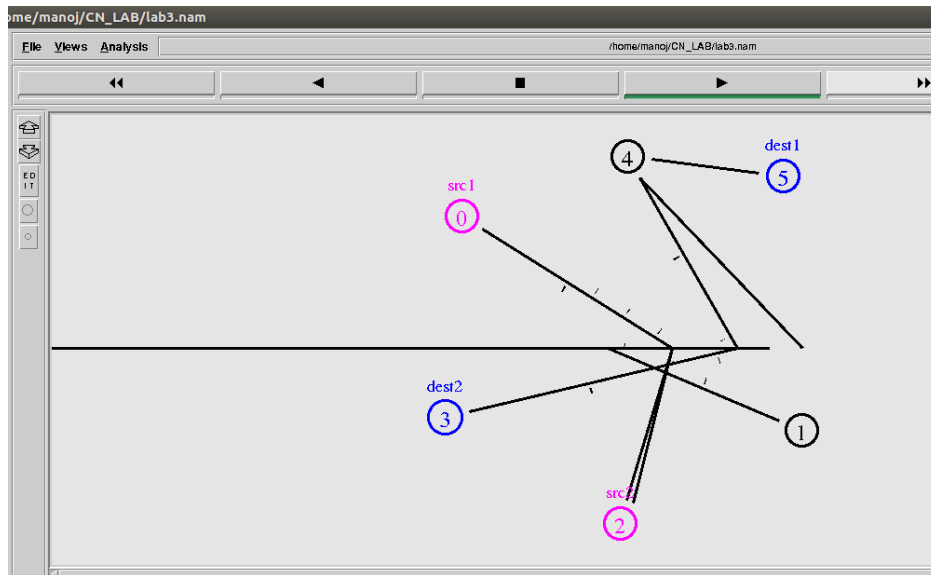
```
if($6=="cwnd_")
```

```
printf("%f\t%f\t\n",$1,$7);
```

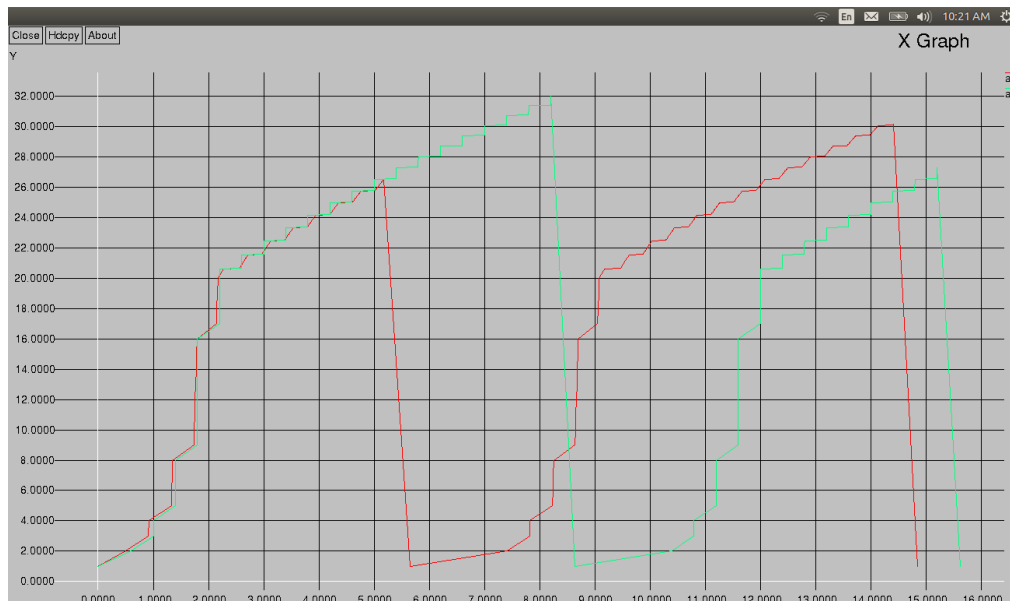
```
}
```

```
END {
```

```
}
```



## 1.7 Results & Analysis



## 1.8 Outcome & Conclusion

- When the congestion window is below the threshold, the congestion window grows exponentially.
- When the congestion window is above the threshold, the congestion window grows linearly.



- Whenever there is a timeout, the threshold is set to one half of the current congestion window and the congestion window is then set to one.

#### **1.9 Remarks**

- TCP congestion control mechanism is observed.

Faculty Signature

#### **1.0 Experiment**

Experiment No	Date Planed	Date Conducted	Marks
04			

**TITLE:**

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

#### **1.1 Learning Objectives**

- To understand the simulation of wire less LAN.

#### **1.2 Aim**



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

### **1.3 Material / Equipment Required**

### **1.4 Theory / Hypothesis**

A wireless network is very similar to the wired network in that all the same pieces are still required, a wireless NIC, Access Point (similar to a wired network Ethernet Switch) and a wireless router. The only thing that's missing is the cables.

First you will need to determine if your desktop or Laptop PC already has a wireless NIC built in. Again, check your users guide to confirm this. Once your wireless NIC is installed you will need to configure the SSID (service set identifier) on each of the PC's to use the same name. Also, you need to make sure all the wireless NICs are configured to be on the same channel and set levels of encryption if desired. Your wireless access point or router should come with detailed explanation of how to configure the SSID and security features such as encryption and access lists. Many of today's wireless routers come with the access point functionality built right in.





## 1.5 Procedure / Program / Activity

### Program:

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

set topo [new Topography]
$topo load_flatgrid 1000 1000

set nf [open lab4.nam w]
$ns namtrace-all-wireless $nf 1000 1000
$ns node-config -adhocRouting DSDV \
    -llType LL \
    -macType Mac/802_11 \
    -ifqType Queue/DropTail \
    -ifqLen 50 \
    -phyType Phy/WirelessPhy \
    -channelType Channel/WirelessChannel \
    -propType Propagation/TwoRayGround \
    -antType Antenna/OmniAntenna \
    -topoInstance $topo \
    -agentTrace ON \
    -routerTrace ON
create-god 3

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

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

$n0 set X_ 50
$n0 set Y_ 50
$n0 set Z_ 0

$n1 set X_ 100
$n1 set Y_ 100
$n1 set Z_ 0

$n2 set X_ 600
$n2 set Y_ 600
$n2 set Z_ 0
```



```
$ns at 0.1 "$n0 setdest 50 50 15"  
$ns at 0.1 "$n1 setdest 100 100 25"  
$ns at 0.1 "$n2 setdest 600 600 25"
```

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

```
set ftp0 [new Application/FTP]  
$ftp0 attach-agent $tcp0  
set sink1 [new Agent/TCPSink]  
$ns attach-agent $n1 $sink1  
$ns connect $tcp0 $sink1  
set tcp1 [new Agent/TCP]  
$ns attach-agent $n1 $tcp1  
set ftp1 [new Application/FTP]  
$ftp1 attach-agent $tcp1  
set sink2 [new Agent/TCPSink]  
$ns attach-agent $n2 $sink2  
$ns connect $tcp1 $sink2  
$ns at 5 "$ftp0 start"  
$ns at 5 "$ftp1 start"  
$ns at 100 "$n1 setdest 550 550 15"  
$ns at 190 "$n1 setdest 70 70 15"  
proc finish { } {  
    global ns nf tf  
    $ns flush-trace  
    exec nam lab4.nam &  
    close $tf  
    exit 0  
}  
$ns at 250 "finish"  
$ns run
```

AWK File:

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

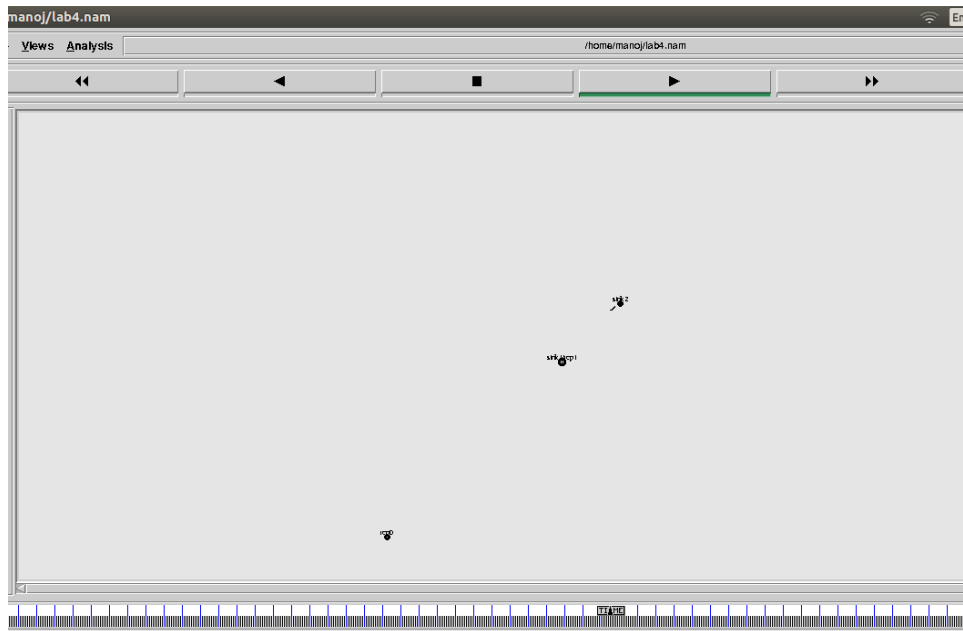


```
{
count1++;
pack1=pack1+$8 ;
time1=$2 ;
}
if($1=="r" && $3=="_2_" && $4=="AGT")
{
count2++;
pack2=pack2+$8 ;
time2=$2 ;
}
}
END{
printf("The Throughput from n0 to n1: %f Mbps \n",
((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps",
((count2*pack2*8)/(time2*1000000)));
}
```

Procedure:

- 1) Open vi editor and type program. Program name should have the extension “.tcl”  
**[root@localhost ~]# vi lab4.tcl**
- 2) Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press **Enterkey**.
- 3) Open vi editor and type **awk** program. Program name should have the extension “.awk”  
**[root@localhost ~]# vi lab4.awk**
- 4) Save the program by pressing “ESC key” first, followed by “Shift and :” keys simultaneously and type “wq” and press **Enterkey**.
- 5) Run the simulation program  
**[root@localhost~]# ns lab4.tcl**
  - i) Here “ns” indicates network simulator. We get the topology shown in the snapshot.
  - ii) Now press the play button in the simulation window and the simulation will begin.
- 6) After simulation is completed run **awk file** to see the output,  
**[root@localhost~]# awk -f lab4.awk lab4.tr**
- 7) To see the trace file contents open the file as,  
**[root@localhost~]# vi lab4.tr**

## **1.6 Block / Circuit / Model / Reaction Diagram**



## 1.7 Results & Analysis

```
Terminal File Edit View Search Terminal Help
manoj@manoj-Inspiron-N5110:~$ ns expt4.tcl
warning: Please use -channel as shown in tcl/ex/wireless-mitf.tcl
num_nodes is set 3
INITIALIZE THE LIST xListHead
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5, distCST_ = 550.0
SORTING LISTS ...DONE!
manoj@manoj-Inspiron-N5110:~$ awk -f expt4.awk lab4.tr
The Throughput from n0 to n1: 5863.442245 Mbps
The Throughput from n1 to n2: 1307.611834 Mbpsmanoj@manoj-Inspiron-N5110:~$
```

## 1.8 Outcome & Conclusion

- An **extended service set (ESS)** is a set of two or more interconnected wireless BSSs that share the same SSID (network name), security credentials and integrated wired local area networks that appear as a single BSS to the logical link control layer at any station associated with one of those BSSs. This facilitates mobile IP and fast secure roaming applications.





### **1.9 Remarks**

- Performance depends on node's mobility.

Faculty Signature

### **1.0 Experiment**



Experiment No	Date Planed	Date Conducted	Marks
05			

**TITLE:**

Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment

**1.1 Learning Objectives**

- **Working principle of GSM using NS2.**

**1.2 Aim**

- Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment

**1.3 Material / Equipment Required**

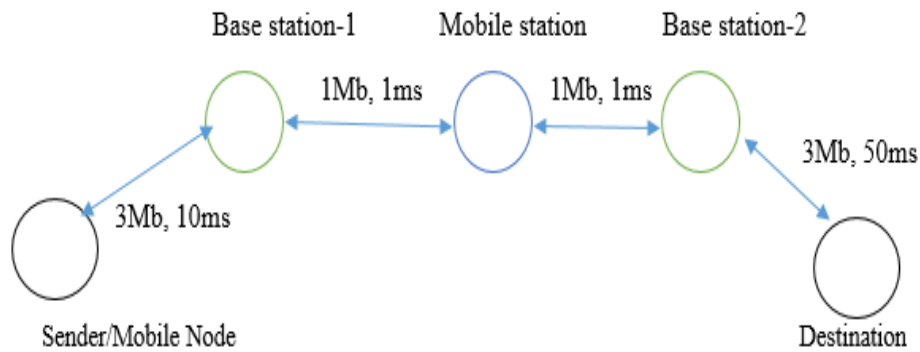
**1.4 Theory / Hypothesis**

Second Generation (2G) technology is based on the technology known as global system for mobile communication (GSM). This technology enabled various networks to provide services like text messages, picture messages and MMS. The technologies used in 2G are either TDMA (Time Division Multiple Access) which divides signal into different time slots or CDMA (Code Division Multiple Access) which allocates a special code to each user so as to communicate over a multiplex physical channel.

GSM uses a variation of time division multiple access (TDMA). 2G networks developed as a replacement for first generation (1G) analog cellular networks, and the GSM standard originally described as a digital, circuit-switched network optimized for full duplex voice telephony. This expanded over time to include data communications, first by circuit-switched transport, then by packet data transport via GPRS (General Packet Radio Services).

GSM can be implemented on all the versions of NS2 (Since year 2004: ns-2.27, and later versions of NS2)

**Design:**



### 1.5 Procedure / Program / Activity

#### Program:

```
# General Parameters
set stop 100 ;# Stop time.
# Topology
set type gsm ;#type of link:
# AQM parameters
set minth 30 ;
set maxth 0 ;
set adaptive 1 ;# 1 for Adaptive RED, 0 for plain RED
# Traffic generation.
set flows 0 ;# number of long-lived TCP flows
set window 30 ;# window for long-lived traffic
set web 2 ;# number of web sessions
# Plotting statics.
set opt(wrap) 100 ;# wrap plots?
set opt(srcTrace) is ;# where to plot traffic
set opt(dstTrace) bs2 ;# where to plot traffic

#default downlink bandwidth in bps
set bwDL(gsm) 9600
#default uplink bandwidth in bps
set bwUL(gsm) 9600
#default downlink propagation delay in seconds
set propDL(gsm) .500
#default uplink propagation delay in seconds
set propUL(gsm) .500

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



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

```
set nodes(is) [$ns node]
set nodes(ms) [$ns node]
$nodes(ms) label "Mobile Station"
set nodes(bs1) [$ns node]
set nodes(bs2) [$ns node]
set nodes(lp) [$ns node]
```

```
proc cell_topo { } {
    global ns nodes
    $ns duplex-link $nodes(lp) $nodes(bs1) 3Mbps 10nodes(ms) DropTail
    $ns duplex-link $nodes(bs1) $nodes(ms) 1 1 RED
    $ns duplex-link $nodes(ms) $nodes(bs2) 1 1 RED
    $ns duplex-link $nodes(bs2) $nodes(is) 3Mbps 10nodes(ms) DropTail
    puts " GSM Cell Topology"
}
```

```
proc set_link_para {t} {
    global ns nodes bwUL bwDL propUL propDL buf
    $ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) duplex
    $ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) duplex
    $ns delay $nodes(bs1) $nodes(ms) $propDL($t) duplex
    $ns delay $nodes(bs2) $nodes(ms) $propDL($t) duplex
    $ns queue-limit $nodes(bs1) $nodes(ms) 10
    $ns queue-limit $nodes(bs2) $nodes(ms) 10
}
```

```
# RED and TCP parameters
Queue/RED set adaptive_ $adaptive
Queue/RED set thresh_ $minth
Queue/RED set maxthresh_ $maxth
Agent/TCP set window_ $window
```

```
source ns-allinone-2.35/ns-2.35/tcl/ex/wireless-scripts/web.tcl
```

```
#Create topology
switch $type {
    gsm -
    gprs -
    umts { cell_topo }
}
set_link_para $type
$ns insert-delayer $nodes(ms) $nodes(bs1) [new Delayer]
$ns insert-delayer $nodes(ms) $nodes(bs2) [new Delayer]
```



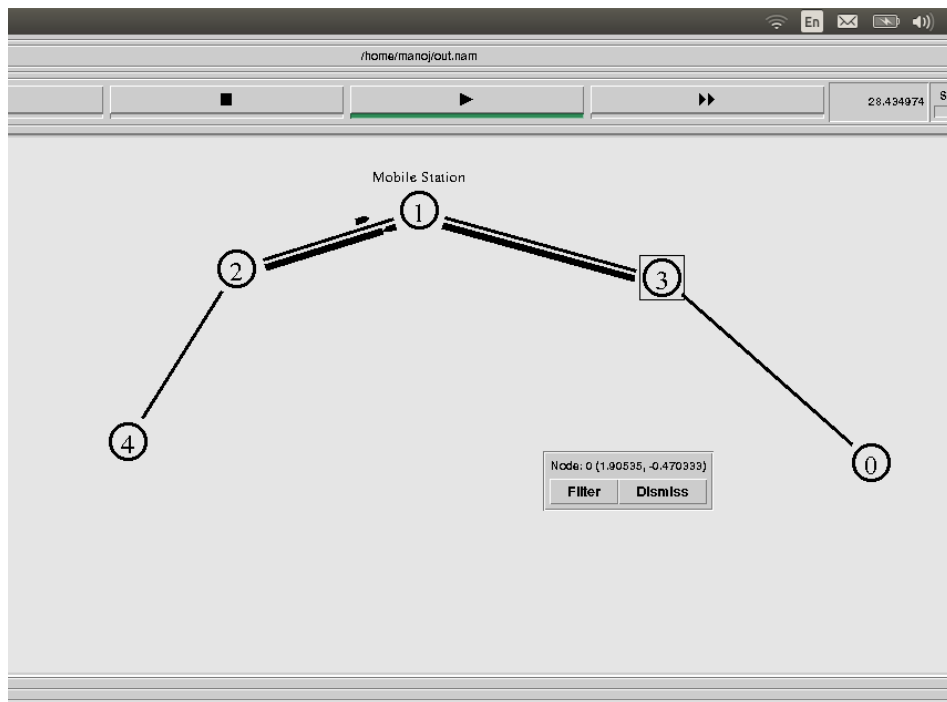
```
# Set up forward TCP connection
if { $flows == 0 } {
    set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1
$nodes(lp) 0]
    set ftp1 [[set tcp1] attach-app FTP]
    $ns at 0.8 "[set ftp1] start"
}
if { $flows > 0 } {
    set tcp1 [$ns create-connection TCP/Sack1 $nodes(is)
TCPSink/Sack1 $nodes(lp) 0]
    set ftp1 [[set tcp1] attach-app FTP]
    $tcp1 set window_ 100
    $ns at 0.0 "[set ftp1] start"
    $ns at 3.5 "[set ftp1] stop"
    set tcp2 [$ns create-connection TCP/Sack1 $nodes(is)
TCPSink/Sack1 $nodes(lp) 0]
    set ftp2 [[set tcp2] attach-app FTP]
    $tcp2 set window_ 3
    $ns at 1.0 "[set ftp2] start"
    $ns at 8.0 "[set ftp2] stop"
}

proc stop {} {
    global nodes opt namf
    set wrap $opt(wrap)
    set sid [$nodes($opt(srcTrace)) id]
    set did [$nodes($opt(dstTrace)) id]
    set a "out.tr"

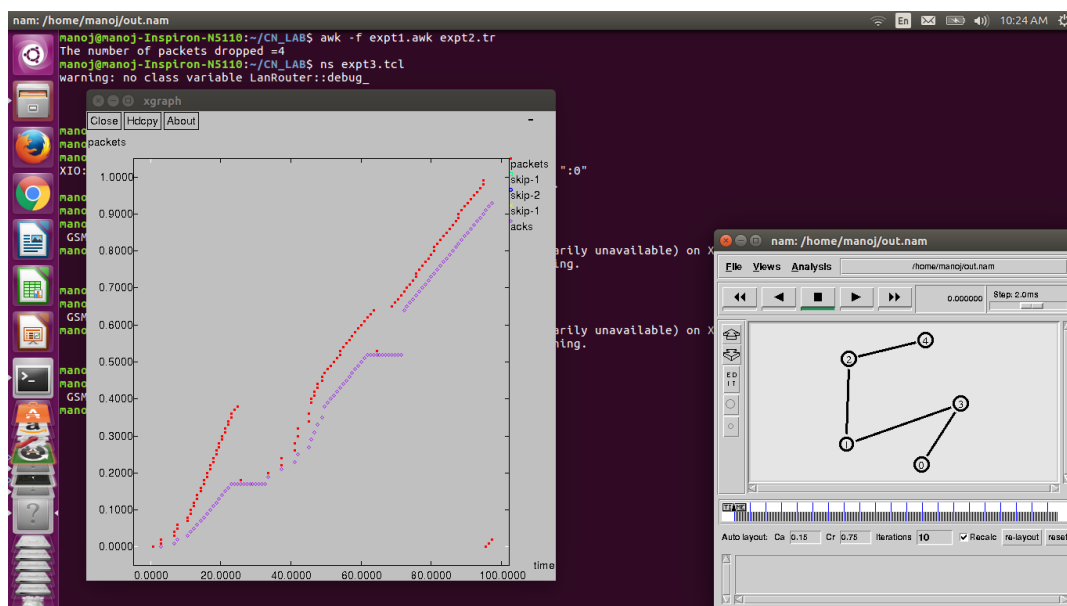
    set GETRC "ns-allinone-2.35/ns-2.35/bin/getrc"
    set RAW2XG "ns-allinone-2.35/ns-2.35/bin/raw2xg"
    exec $GETRC -s $sid -d $did -f 0 out.tr | \
$RAW2XG -s 0.01 -m $wrap -r > plot.xgr
    exec $GETRC -s $did -d $sid -f 0 out.tr | \
$RAW2XG -a -s 0.01 -m $wrap >> plot.xgr
    exec nam out.nam &
    exec xgraph -x time -y packets plot.xgr &
    exit 0
}
$ns at $stop "stop"
$ns run
```

Procedure:

## **1.6 Block / Circuit / Model / Reaction Diagram**



## 1.7 Results & Analysis



## 1.8 Outcome & Conclusion

- .performance of a network is measured with respect to the number of packets transmitted and the number of acks received.



### 1.9 Remarks

- The number of packets transmitted and acks received is observed.

Faculty Signature

### 1.0 Experiment

Experiment No	Date Planed	Date Conducted	Marks
06			

#### TITLE:

**Implement and study the performance of CDMA on NS2/NS3 (UsingstackcalledCall net) or equivalent environment.**

### 1.1 Learning Objectives

- Working principle of 3G networks using NS2.

### 1.2 Aim

Implement and study the performance of CDMA on NS2/NS3 (UsingstackcalledCall net) or equivalent environment.

### 1.3 Material / Equipment Required

### 1.4 Theory / Hypothesis

3G networks developed as a replacement for second generation (2G) GSM standard networkwithfullduplexvoicetelephony.CDMAisusedastheaccessmethodinmanymobilephone

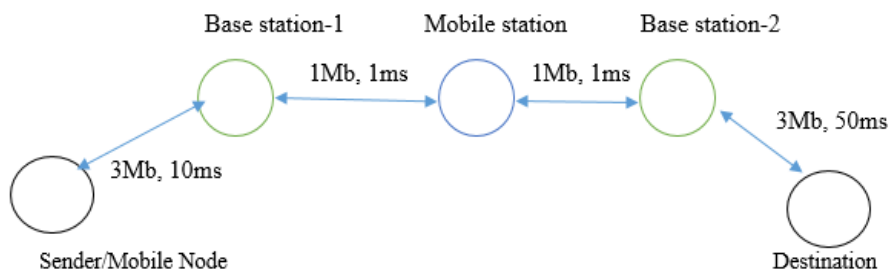


standards. IS-95, also called cdmaOne, and its 3G evolution CDMA2000, are often simply referred to as CDMA, but UMTS (The Universal Mobile Telecommunications System is a third generation mobile cellular system for networks based on the GSM standard.), the 3G standard used by GSM carriers, also uses wideband CDMA. Long-Term Evolution (LTE) is a standard for high-speed wireless communication which uses CDMA network technology.

3G technology generally refers to the standard of accessibility and speed of mobile devices. The standards of the technology were set by the International Telecommunication Union (ITU). This technology enables use of various services like GPS (Global Positioning System), mobile television and video conferencing. It not only enables them to be used worldwide, but also provides with better bandwidth and increased speed. The main aim of this technology is to allow much better coverage and growth with minimum investment.

CDMA can be implemented on all the versions of NS2 (Since year 2004: ns-2.27, and later versions of NS2)

### Design:



## 1.5 Procedure / Program / Activity

Program:

```
# General Parameters
set stop 100 ;# Stop time.
# Topology
set type cdma ;#type of link:
# AQM parameters
set minth 30 ;
set maxth 0 ;
set adaptive 1 ;# 1 for Adaptive RED, 0 for plain RED
# Traffic generation.
set flows 0 ;# number of long-lived TCP flows
set window 30 ;# window for long-lived traffic
set web 2 ;# number of web sessions
# Plotting statics.
set opt(wrap) 100 ;# wrap plots?
set opt(srcTrace) is ;# where to plot traffic
set opt(dstTrace) bs2 ;# where to plot traffic
```





```
#default downlink bandwidth in bps
set bwDL(cdma) 384000
#default uplink bandwidth in bps
set bwUL(cdma) 64000
#default downlink propagation delay in seconds
set propDL(cdma) .150
#default uplink propagation delay in seconds
set propUL(cdma) .150

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

set nodes(is) [$ns node]
set nodes(ms) [$ns node]
set nodes(bs1) [$ns node]
set nodes(bs2) [$ns node]
set nodes(lp) [$ns node]

proc cell_topo { } {
    global ns nodes
    $ns duplex-link $nodes(lp) $nodes(bs1) 3Mbps 10nodes(ms) DropTail
    $ns duplex-link $nodes(bs1) $nodes(ms) 1 1 RED
    $ns duplex-link $nodes(ms) $nodes(bs2) 1 1 RED
    $ns duplex-link $nodes(bs2) $nodes(is) 3Mbps 50nodes(ms) DropTail
    puts " cdma Cell Topology"
}

proc set_link_para {t} {
    global ns nodes bwUL bwDL propUL propDL buf
    $ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) duplex
    $ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) duplex
    $ns delay $nodes(bs1) $nodes(ms) $propDL($t) duplex
    $ns delay $nodes(bs2) $nodes(ms) $propDL($t) duplex
    $ns queue-limit $nodes(bs1) $nodes(ms) 20
    $ns queue-limit $nodes(bs2) $nodes(ms) 20
}

# RED and TCP parameters
Queue/RED set adaptive_ $adaptive
Queue/RED set thresh_ $minth
Queue/RED set maxthresh_ $maxth
Agent/TCP set window_ $window

source ns-allinone-2.35/ns-2.35/tcl/ex/wireless-scripts/web.tcl

#Create topology
```



```
switch $type {

cdma { cell_topo}
}
set_link_para $type
$ns insert-delayer $nodes(ms) $nodes(bs1) [new Delayer]
$ns insert-delayer $nodes(ms) $nodes(bs2) [new Delayer]

# Set up forward TCP connection
if {$flows == 0} {
    set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1
$nodes(lp) 0]
    set ftp1 [[set tcp1] attach-app FTP]
    $ns at 0.8 "[set ftp1] start"
}
if {$flows > 0} {
    set tcp1 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp)
0]
    set ftp1 [[set tcp1] attach-app FTP]
    $tcp1 set window_ 100
    $ns at 0.0 "[set ftp1] start"
    $ns at 3.5 "[set ftp1] stop"
    set tcp2 [$ns create-connection TCP/Sack1 $nodes(is) TCPSink/Sack1 $nodes(lp)
0]
    set ftp2 [[set tcp2] attach-app FTP]
    $tcp2 set window_ 3
    $ns at 1.0 "[set ftp2] start"
    $ns at 8.0 "[set ftp2] stop"
}

proc stop {} {
    global nodes opt nf
    set wrap $opt(wrap)
    set sid [$nodes($opt(srcTrace)) id]
    set did [$nodes($opt(dstTrace)) id]
    set a "out.tr"
    set GETRC "ns-allinone-2.35/ns-2.35/bin/getrc"
    set RAW2XG "ns-allinone-2.35/ns-2.35/bin/raw2xg"
    exec $GETRC -s $sid -d $did -f 0 out.tr | \
    $RAW2XG -s 0.01 -m $wrap -r > plot.xgr
    exec $GETRC -s $did -d $sid -f 0 out.tr | \
    $RAW2XG -a -s 0.01 -m $wrap >> plot.xgr
    exec xgraph -x time -y packets plot.xgr &
    exit 0
}
$ns at $stop "stop"
```



## 1.6 Block / Circuit / Model / Reaction Diagram

The screenshot shows a Linux desktop environment with a purple background. On the left is a vertical dock with various application icons. The main area contains two windows:

- Terminal Window:** Displays the following commands and output:
 

```
nano@manoj-Inspiron-N5110:~/CN_LAB$ awk -f expt1.awk expt2.tr
The number of packets dropped =4
nano@manoj-Inspiron-N5110:~/CN_LAB$ ns expt3.tcl
warning: no class variable LanRouter::debug_
```
- xgraph Window:** A window titled "xgraph" with a menu bar (Close, Help, About). It displays a line graph with "packets" on the y-axis (0.0000 to 1.0000) and "time" on the x-axis (0.0000 to 100.0000). The graph shows a red line that starts at (0,0), rises sharply to 1.0000 by approximately time 10, and then remains constant at 1.0000 for the rest of the duration. The legend on the right lists:
  - packets
  - skip-1
  - skip-2
  - ack-1
  - acks

On the right side of the terminal window, there are several error messages:
 

```
...:0"
...
...rily unavailable) on X server ":0"
...ing.
...
...rily unavailable) on X server ":0"
...ing.
...
...rily unavailable) on X server ":0"
...
...rily unavailable) on X server ":0"
```

- performance of a network is measured with respect to the number of packets transmitted and the number of acks received.

- The number of packets transmitted and acks received is observed.

## 1.0 Experiment

Page 35



**TITLE:**

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

**1.1 Learning Objectives**

- To understand error detection algorithms.
- To understand working principle of CRC-CCITT (16- bits)

**1.2 Aim**

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

**1.3 Material / Equipment Required**

- Intel based PC
- 

**1.4 Theory / Hypothesis**

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

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

$$\begin{array}{r} 101 = 5 \\ \hline 10011 / 1101101 \\ \underline{10011} \phantom{00} \\ 00000 \\ \underline{00000} \\ 00000 \\ \underline{00000} \\ 00000 \\ \underline{00000} \\ 00000 \\ \underline{00000} \\ 00000 \end{array}$$

1110 = 14 = remainder



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

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

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

### 1.5 Procedure / Program / Activity

#### • PROGRAM

### 1.6 Formula / Calculations

Let frame=101  
 10001000000100001 ) 101000000000000000  
     10001000000100001  
       0010100000010000100  
     10001000000100001  
       00101000010100101  
 Check sum is 0101000010100101

case 1: Let Received codeword is 1010101000010100101  
 10001000000100001 ) 1010101000010100101  
     10001000000100001  
       0010001000000100001  
     10001000000100001  
       00000000000000000  
 Data is received with no error

case 2: Let Received codeword is 1010101000010100111  
 10001000000100001 ) 1010101000010100111  
     10001000000100001



0010001000000100011  
10001000000100001  
00000000000000010  
Data is received with error

## **1.7 Results & Analysis**

## **1.8 Outcome & Conclusion**

- Perform division operation on received codeword by using generator and check the remainder.
- If the remainder is not equal to zero at the destination then received codeword is corrupted.
- If the remainder is equal to zero then either the data is not corrupted or the data may be corrupted but not able to detect it.

## **1.9 Remarks**

- Errors can be detected by using CRC.

Faculty Signature



## 1.0 Experiment

Experiment No	Date Planed	Date Conducted	Marks
08			

### **TITLE:**

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

## 1.1 Learning Objectives

- To understand the routing mechanism.

## 1.2 Aim

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

## 1.3 Material / Equipment Required

## 1.4 Theory / Hypothesis

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

Routers that use this algorithm have to maintain the distance tables (which is a one-dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always updated by exchanging information with the neighboring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.

The 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. 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. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e. a cycle whose edge sum to a negative value) that is reachable from the source, then there is no



cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman–Ford algorithm can detect negative cycles and report their existence

#### **1.5 Procedure / Program / Activity**

- **PROGRAM**

[

#### **1.6 Results & Analysis**

#### **1.7 Outcome & Conclusion**

- Shortest path between source and destination can be computed by using Distance Vector algorithm which uses distributed bellman ford concept.

#### **1.8 Remarks**

- Shortest path between source and destination can be computed.

Faculty Signature





## **1.0 Experiment**

Experiment No	Date Planed	Date Conducted	Marks
09			

### **TITLE:**

Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

## **1.1 Learning Objectives**

- **To Understand socket programming with TCP.**

## **1.2 Aim**

- Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

## **1.3 Material / Equipment Required**

## **1.4 Theory / Hypothesis**

With the server process running, the client process can initiate a TCP connection to the server. This is done in the client program by creating a socket object. When the client creates its socket object, it specifies the address of the server process, namely, the IP address of the server and the port number of the process. Upon creation of the socket object, TCP in the client initiates a three-way handshake and establishes a TCP connection with the server. The three-way handshake is completely transparent to the client and server programs.

During the three-way handshake, the client process knocks on the welcoming door of the server process. When the server "hears" the knocking, it creates a new door (i.e., a new socket) that is dedicated to that particular client. At the end of the handshaking phase, a TCP connection exists between the client's socket and the server's new socket. Henceforth, we refer to the new socket as the server's "connection socket".

## **1.5 Procedure / Program / Activity**



- **PROGRAM**

**/\*TCP Client\*/**

**package exp1;**

**import java.io.\*;**  
**import java.net.\*;**  
**import java.util.Scanner;**

**public class TCPClient**  
**{**

**public static void main(String[] args) throws**  
**IOException, InterruptedException**

**{**

**DataOutputStream out;**  
**DataInputStream in;**  
**Scanner scanner = new Scanner(System.in);**

**Socket socket = new Socket("127.0.0.1", 6000);**

**System.out.println("Client Connected to Server");**  
**System.out.print("\nEnter the filename to request\n");**  
**String filename = scanner.nextLine();**

**in = new DataInputStream(socket.getInputStream());**  
**out = new DataOutputStream(socket.getOutputStream());**

**out.writeUTF(filename);**

**String fileContent = in.readUTF();**

**if (fileContent.length() > 0)**  
**System.out.println(fileContent);**

**else**  
**System.out.println("FILE IS EMPTY");**

**}**

**}**

**/\*TCP SERVER\*/**

**package exp1;**

**import java.io.\*;**  
**import java.net.\*;**



```
import java.nio.file.*;

public class TCPServer
{

    public static void main(String[] args) throws IOException
    {
        ServerSocket server;
        DataOutputStream out = null;
        DataInputStream in;

        try
        {
            server = new ServerSocket(6000, 1);
            System.out.println("Server Waiting for client");
            Socket socket = server.accept();

            System.out.println("Client connected ");

            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());

            String fileName = in.readUTF();
            System.out.println("File Requested is : " + fileName);
            byte[] filedata = Files.readAllBytes(Paths.get(fileName));
            String fileContent = new String(filedata);

            out.writeUTF(fileContent.toString());
            System.out.println("FILE SENT SUCCESSFULLY");
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
            out.writeUTF("FILE DOESN'T EXISTS");
        }
    }
}
```

## **1.6 Results & Analysis**



### 1.7 Outcome & Conclusion

- A *socket* is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.
- An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints.
- Client process sends the request and the server provides services to these requests.

### 1.8 Remarks

- Server process sends the content of requested file to the client process by using TCP/IP sockets.

**FACULTY SIGNATURE**

### 1.0 Experiment

Experiment No	Date Planed	Date Conducted	Marks
10			

**TITLE:**

**Write a program on datagram socket for client/server to display the messages on client side, typed at the serverside.**

### 1.1 Learning Objectives

- To understand socket programming with UDP.

### 1.2 Aim



- Write a program on datagram socket for client/server to display the messages on client side, typed at the serverside.

### **1.3 Material / Equipment Required**

### **1.4 Theory / Hypothesis**

UDP also allows two (or more) processes running on different hosts to communicate. However, UDP differs from TCP in many fundamental ways. First, UDP is a connectionless service -- there isn't an initial handshaking phase during which a pipe is established between the two processes. Because UDP doesn't have a pipe, when a process wants to send a batch of bytes to another process, the sending process must include attach the destination process's address to the batch of bytes. And this must be done for each batch of bytes the sending process sends. As with TCP, the destination address is a tuple consisting of the IP address of the destination host and the port number of the destination process. We shall refer to the batch of information bytes along with the IP destination address and port number as the "packet". After having created a packet, the sending process pushes the packet into the network through a socket. UDP provides an unreliable transport service to its communication processes – it makes no guarantees that a datagram will reach its ultimate destination.

### **1.5 Procedure / Program / Activity**

- **PROGRAM**

**/\* UDP CLIENT\*/**

```
package exp1;  
import java.io.*;  
import java.net.*;
```

```
public class UDPclient {
```

```
    public static void main(String[] args) throws Exception  
    {
```

```
        BufferedReader cin= new BufferedReader(new  
        InputStreamReader(System.in));  
        DatagramSocket ClientSocket=new DatagramSocket();  
        InetAddress IPAddress=InetAddress.getByName("192.168.200.20");  
        byte[] sendData=new byte[1024];  
        byte[] receiveData= new byte[1024];
```

```
        String sentence= cin.readLine();
```



```
        sendData=sentence.getBytes();

        DatagramPacket sendPacket= new
        DatagramPacket(sendData,sendData.length,IPAddress,8000);
        ClientSocket.send(sendPacket);

        DatagramPacket receivePacket=new
        DatagramPacket(receiveData,receiveData.length);
        ClientSocket.receive(receivePacket);

        String RCVString=new String(receivePacket.getData());
        System.out.println("received from Server"+RCVString);
        ClientSocket.close();

    }
}

/* UDP SERVER*/

package exp1;
import java.io.*;
import java.net.*;

public class UDPServer {

    public static void main(String[] args) throws Exception {
        // TODO Auto-generated method stub
        BufferedReader Cin= new BufferedReader(new
        InputStreamReader(System.in));
        DatagramSocket ServerSocket=new DatagramSocket(8000);

        byte[] receiveData=new byte[1024];
        byte[] sendData=new byte[1024];
        while(true)
        {
            DatagramPacket receivePacket=new
            DatagramPacket(receiveData,receiveData.length);
            ServerSocket.receive(receivePacket);
            String RCVString=new String(receivePacket.getData());
            System.out.println("received from Client "+RCVString);
            InetAddress ClientAddress= receivePacket.getAddress();
            int port=receivePacket.getPort();

            String Sentence=Cin.readLine();
            sendData=Sentence.getBytes();
```



```
        DatagramPacket sendPacket=new
        DatagramPacket(sendData,sendData.length,ClientAddress,port);
        ServerSocket.send(sendPacket);
    }
}
```

## **1.6 Results & Analysis**

## **1.7 Outcome & Conclusion**

- Client process and server process communicate with each other by exchanging the messages.

## **1.8 Remarks**

- Client process and Server process communicate with each other by using datagram sockets.

**FACULTY SIGNATURE**

## **1.0 Experiment**

Experiment No	Date Planed	Date Conducted	Marks
11			

**TITLE:**

**Write a program for simple RSA algorithm to encrypt and decrypt the data.**

## **1.1 Learning Objectives**

- To understand the Implementation of security algorithm.

## **1.2 Aim**



**Write a program for simple RSA algorithm to encrypt and decrypt the data.**

### **1.3 Material / Equipment Required**

### **1.4 Theory / Hypothesis**

RSA is an example of public key cryptography. It was developed by Rivest, Shamir and Adelman. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

The RSA algorithm's efficiency requires a fast method for performing the modular exponentiation operation. A less efficient, conventional method includes raising a number (the input) to a power (the secret or public key of the algorithm, denoted  $e$  and  $d$ , respectively) and taking the remainder of the division with  $N$ . A straight-forward implementation performs these two steps of the operation sequentially: first, raise it to the power and second, apply modulo. The RSA algorithm comprises of three steps, which are depicted below:

#### **Key Generation Algorithm**

1. Generate two large random primes,  $p$  and  $q$ , of approximately equal size such that their product  $n = p * q$
2. Compute  $n = p * q$  and Euler's totient function  $(\phi)$   $\phi(n) = (p-1)(q-1)$ .
3. Choose an integer  $e$ ,  $1 < e < \phi$ , such that  $\gcd(e, \phi) = 1$ .
4. Compute the secret exponent  $d$ ,  $1 < d < \phi$ , such that  $e * d \equiv 1 \pmod{\phi}$ .
5. The public key is  $(e, n)$  and the private key is  $(d, n)$ . The values of  $p$ ,  $q$ , and  $\phi$  should also be kept secret.

#### **Encryption**

Sender A does the following:-

1. Using the public key  $(e, n)$
2. Represents the plaintext message as a positive integer  $M$
3. Computes the cipher text  $C = M^e \pmod{n}$ .
4. Sends the cipher text  $C$  to B (Receiver).

#### **Decryption**

Recipient B does the following:-

1. Uses his private key  $(d, n)$  to compute  $M = C^d \pmod{n}$ .
2. Extracts the plaintext from the integer representation  $M$ .





## 1.5 Procedure / Program / Activity

### Program:

```
package exp1;
import java.util.Scanner;

class RSA
{
    public static void main(String args[])
    {
        Scanner ip=new Scanner(System.in);
        int p,q,n,e=1,j,z,d,i,k;
        int slen;
        int pt[]= new int[10];
        int ct[]= new int[10];
        int dt[]= new int[10];

        String s=new String();
        System.out.println("Enter the two prime numbers:");
        p=ip.nextInt();
        q=ip.nextInt();
        System.out.print("Enter the message to be sent");
        s=ip.next();

        slen=s.length();
        for(j=0;j<slen;j++)
        {
            pt[j]=(int)(s.charAt(j));
        }
        n=p*q;
        z=(p-1)*(q-1);

        for(i=1;i<z;i++)
        {
            if(z%i==0)
                continue;
            else
                break;
        }
        e=i;
        System.out.println("encryption key"+n+" "+e);
        for(i=1;i<z;i++)
            if(((e*i-1)%z)==0)
```



```
        break;
    d=i;

    System.out.println("decryption key"+n+" "+d);

    for(j=0;j<slen;j++)
    {
        k=pt[j];
        for(i=1;i<e;i++)
        {
            k=k*pt[j];
            k=k%n;
        }
        k=k%n;
        ct[j]=k;
    }

    System.out.println("Plaintext ----Cipher Text");
    for(j=0;j<slen;j++)
    {
        System.out.println(s.charAt(j)+" "+ct[j]);
    }

    for(j=0;j<slen;j++)
    {
        k=ct[j];
        for(i=1;i<d;i++)
        {
            k=k*ct[j];
            k=k%n;
        }
        k=k%n;
        dt[j]=k;
    }

    System.out.println("Ciphertext ----PlainText");
    for(j=0;j<slen;j++)
    {
        System.out.println(ct[j]+" "+(char)dt[j]);
    }
    ip.close();
}
```



}

### 1.6 Formula / Calculations

1. Select primes  $P=11$ ,  $Q=3$ .

2.  $N = P \times Q = 11 \times 3 = 33$

$Z = (P-1) \times (Q-1) = 10 \times 2 = 20$

3. Lets choose  $E=3$

Check  $\text{GCD}(E, P-1) = \text{GCD}(3, 10) = 1$  (i.e. 3 and 10 have no common factors except 1),  
and check  $\text{GCD}(E, Q-1) = \text{GCD}(3, 2) = 1$

therefore  $\text{GCD}(E, Z) = \text{GCD}(3, 20) = 1$

4. Compute  $D$  such that  $E \times D \equiv 1 \pmod{Z}$

compute  $D = E^{-1} \pmod{Z} = 3^{-1} \pmod{20}$

find a value for  $D$  such that  $Z$  divides  $((E \times D)-1)$

find  $D$  such that 20 divides  $3D-1$ .

Simple testing ( $D = 1, 2, \dots$ ) gives  $D = 7$

Check:  $(E \times D)-1 = 3 \times 7 - 1 = 20$ , which is divisible by  $Z$ .

5. Public key =  $(N, E) = (33, 3)$

Private key =  $(N, D) = (33, 7)$ .

Now say we want to encrypt the message  $m = 7$ ,

Cipher code =  $M^E \pmod{N}$

$$= 7^3 \pmod{33}$$

$$= 343 \pmod{33}$$

$$= 13.$$

Hence the ciphertext  $c = 13$ .

To check decryption we compute  $\text{Message}' = C^D \pmod{N}$

$$= 13^7 \pmod{33}$$

$$= 7$$

Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that  $a = bc \pmod{n} = (b \pmod{n}) \cdot (c \pmod{n}) \pmod{n}$  so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

### 1.7 Results & Analysis

### 1.8 Outcome & Conclusion

- Source send the data converting the text into ciphertext by using public key of the destination and the destination decrypts the content by using its own private key.

### 1.9 Remarks



- Data can be securely exchanged between source and destination.

## **FACULTY SIGNATURE**

### **1.0 Experiment**

Experiment No	Date Planed	Date Conducted	Marks
12			

#### **TITLE:**

**Write a program for congestion control using leaky bucket algorithm**

### **1.0 Learning Objectives**

- To understand congestion control mechanisms.

### **1.2 Aim**

- **Write a program for congestion control using leaky bucket algorithm**

### **1.3 Material / Equipment Required**

### **1.4 Theory / Hypothesis**

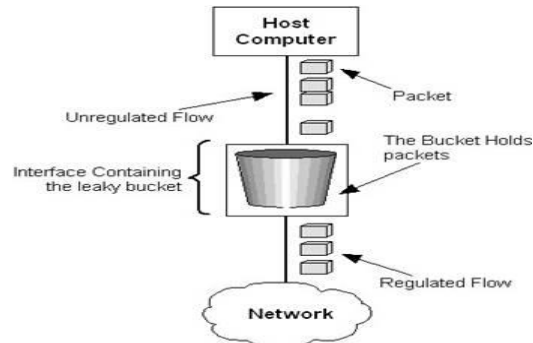
The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spillover. In a similar way if the bucket is empty the output will be zero. From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate



the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).



(a) A leaky bucket with water



(b) A leaky bucket with packets

### 1.5 Procedure / Program / Activity

Program:

```
package exp1;
import java.util.Scanner;
import java.util.Random;
public class LeakyBucket {

    public static void main(String[] args) {
        int n, clk, b_size, o_rate, p_sz_rm=0, p_sz, p_time;
        int[] packets;
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter total number of packets");
        n=sc.nextInt();
        //System.out.println("enter size of n packets");
        packets = new int[n];
        Random rand=new Random();
        for(int i=0;i<n;i++)
        {
            // packets[i]=sc.nextInt();
            packets[i]=(rand.nextInt(5)+1)*10;
        }
        System.out.println("Enter Output rate");
        o_rate=sc.nextInt();
        System.out.println("enter Bucket Size");
        b_size=sc.nextInt();

        for(int i=0;i<n;i++)
```



```
{
    if((packets[i]+p_sz_rm)>b_size)
    {
        if(packets[i]>b_size)
            System.out.println("Incoming packet size" + packets[i]+"is Greater than
                                bucket capacity REJECTED");

        else
            System.out.println("Incomming packet size is" +packets[i] +" Bucket
                                capacity exceeded - REJECTED!!");
    }
    else
    {
        p_sz=packets[i];
        p_sz_rm+=p_sz;
        System.out.println("Incoming Packet size:"+ p_sz);
        System.out.println("Transmission left:"+ p_sz_rm);
        p_time=rand.nextInt(4);

        System.out.println("Next Packet Will come at :"+p_time);
        for(clk=0;clk<=p_time;clk+=1)
        {
            System.out.println("Time Left:" + (p_time-clk));

            if(p_sz_rm!=0)
            {
                if(p_sz_rm<=o_rate)
                {
                    System.out.println(" Bytes Transmitted--"+p_sz_rm);
                    p_sz_rm=0;
                }
                else
                {
                    System.out.println("Bytes Transmitted--"+ o_rate);
                    p_sz_rm-=o_rate;
                }
                System.out.println("Bytes Remaining:--"+p_sz_rm);
            }
            else
                System.out.println("No packets to transmit--");
        }
    }
}

System.out.println();
```



```
sc.close();
```

```
}
```

```
}
```

## **1.6 Results & Analysis**

## **1.7 Outcome & Conclusion**

- Leaky bucket policing device monitor the traffic flow. Whenever bucket overflow occurs , the leaky bucket tag these packets as nonconforming packets and discards the incoming packets and provide service to conformed packets.

## **1.8 Remarks**

- Leaky bucket policing device control the traffic congestion.

**FACULTY SIGNATURE**