DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LESSON PLAN

Name of the Lab : Computer Networks Lab **Subject Code : 18CS602**

Semester : VI B.E **Branch: CSE**

Submitted By: Raghunandan K R, Asst. Professor-GD-II, Dept. of CSE

Prerequisite:

- Sound knowledge in C,Linux
- Basic knowledge in Computer Networks

Objectives:

To provide a good understanding of the salient features of network Programming. Students are expected to understand the network concept in detail and to program using C programming language.

Learning outcome and end use:

By the end of the course, the students should be able to :-

- Have a detailed understanding of the underlying design principles of computers comunication.
- Be able to apply basic concept of TCP/IP protocol and the methods of programming techniqes.
- Recognize the features of client/server systems and programs with a view to be able to implement simple system in this model
- Analyse ,develop and implement error-detection and correction, and encryption algorithms
- Be able to simulate network protocols to check the functionality of the protocols.

Rules of behavior in the laboratory:

- Every laboratory sessions begins SHARP at the specified time in the schedule.
- Each lab session is two hours long. Students are advised to bring their record and observation books.
- Strict discipline should be maintained throughout the lab.
- Punctuality should be followed by each and every student.
- Food, drinks and cell phone are not allowed inside the laboratory

Method of Assessment:

Writing Algorithm	-	10
Program Execution	-	30
Viva Voce	-	10

List of Experiments

Note: Student is required to solve one problem from PART-A and one problem from PART-B.

PART A – Simulation Exercises

The following experiments shall be conducted using either NS2/OPNET or any other simulators.

1. Simulate a 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. Simulate a four node point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets by TCP/UDP.
3. Simulate the different types of Internet traffic such as FTP a TELNET over a network and analyze the throughput.
4. Simulate the transmission of ping messaged over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.
5. Simulate an Ethernet LAN using N-nodes(6-10), change error rate and data rate and compare the throughput.
6. Simulate an Ethernet LAN using N nodes and set multiple traffic nodes and determine collision across different nodes.
7. Simulate an Ethernet LAN using N nodes and set multiple traffic nodes and plot congestion window for different source/destination.

PART B

The following experiments shall be conducted using C/C++.

1. Write a program for error detecting code using CRC-CCITT (16-bits).
2. Write a program for frame sorting technique used in buffers.
3. Write a program for distance vector algorithm to find suitable path for transmission.
4. Write a program for simple RSA algorithm to encrypt and decrypt the data.
5. Write a program for Hamming Code generation for error detection and correction.
6. Write a program for congestion control using Leaky bucket algorithm.

Part A Experiments

1. Overview of NS-2

1.1 Introduction to NS-2

NS-2 is an event driven packet level network simulator developed as part of the VINT project (Virtual Internet Testbed). This was a collaboration of many institutes including UC Berkeley, AT&T, XEROX PARC and ETH. Version 1 of NS was developed in 1995 and with version 2 released in 1996. Version 2 included a scripting language called Object oriented Tcl (OTcl). It is an open source software package available for both Windows and Linux platforms. It provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

NS-2 has many and expanding uses including:

- To evaluate the performance of existing network protocols.
- To evaluate new network protocols before use.
- To run large scale experiments not possible in real experiments.
- To simulate a variety of ip networks

1.2 Downloading/Installing ns

You can download the package from http://www.isi.edu/nsnam/ns/ns-build.html. There are two ways to build ns: from the various packages or ‘all-in-one’ package. For simplicity, it is recommended to start with the ‘all-in-one’ package. Please refer http://www.isi.edu/nsnam/ns/ns-problems.html for any installation problems.

1.3 Starting ns

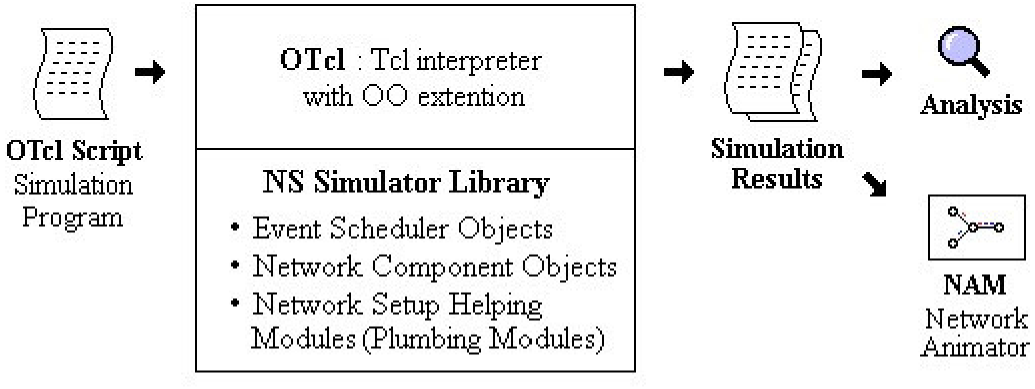
You start ns with the command 'ns <tclscript>' (assuming that you are in the directory with the ns executable, or that your path points to that directory), where '<tclscript>' is the name of a Tcl (Tool Command Language) script file which defines the simulation scenario (i.e. the topology and the events). You can also just start ns without any arguments and enter the Tcl commands in the Tcl shell, but that is definitely less comfortable**.**

1.4 Starting nam (Network Animator)

You can either start nam with the command 'nam <nam-file>' where '<nam-file>' is the name of a nam trace file that was generated by ns, or you can execute it directly out of the Tcl simulation script for the simulation which you want to visualize.

1.5 Architecture of NS-2

As shown in the simplified user's view of Figure, NS is an Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network set-up (plumbing) module libraries.



An OTcl script will do the following.

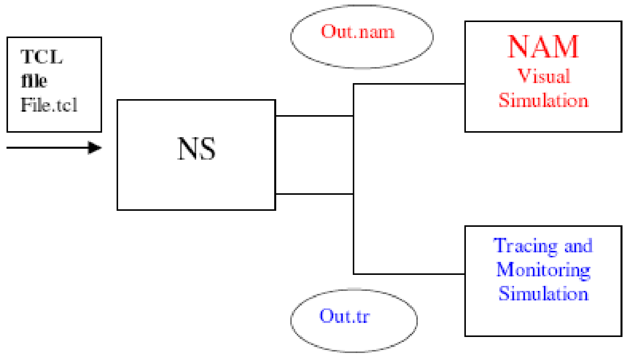
- Initiates an event scheduler.
- Sets up the network topology using the network objects.
- Tells traffic sources when to start/stop transmitting packets through the event scheduler

Another major component of NS besides network objects is the event scheduler. An event in NS is a packet ID that is unique for a packet with scheduled time and the pointer to an object that handles the event. The event scheduler in NS-2 performs the following tasks:

- Organizes the simulation timer.
- Fires events in the event queue.
- Invokes network components in the simulation.

Depending on the user’s purpose for an OTcl simulation script, simulation results are stored as trace files, which can be loaded for analysis by an external application:

1. A NAM trace file (file.nam) for use with the Network Animator Tool
2. A Trace file (file.tr) for use with XGraph or TraceGraph [11].



TclCL is the language used to provide a linkage between C++ and OTcl. Toolkit Command Language (Tcl/OTcl) scripts are written to set up/configure network topologies. TclCL provides linkage for class hierarchy, object instantiation, variable binding and command dispatching. OTcl is used for periodic or triggered events.

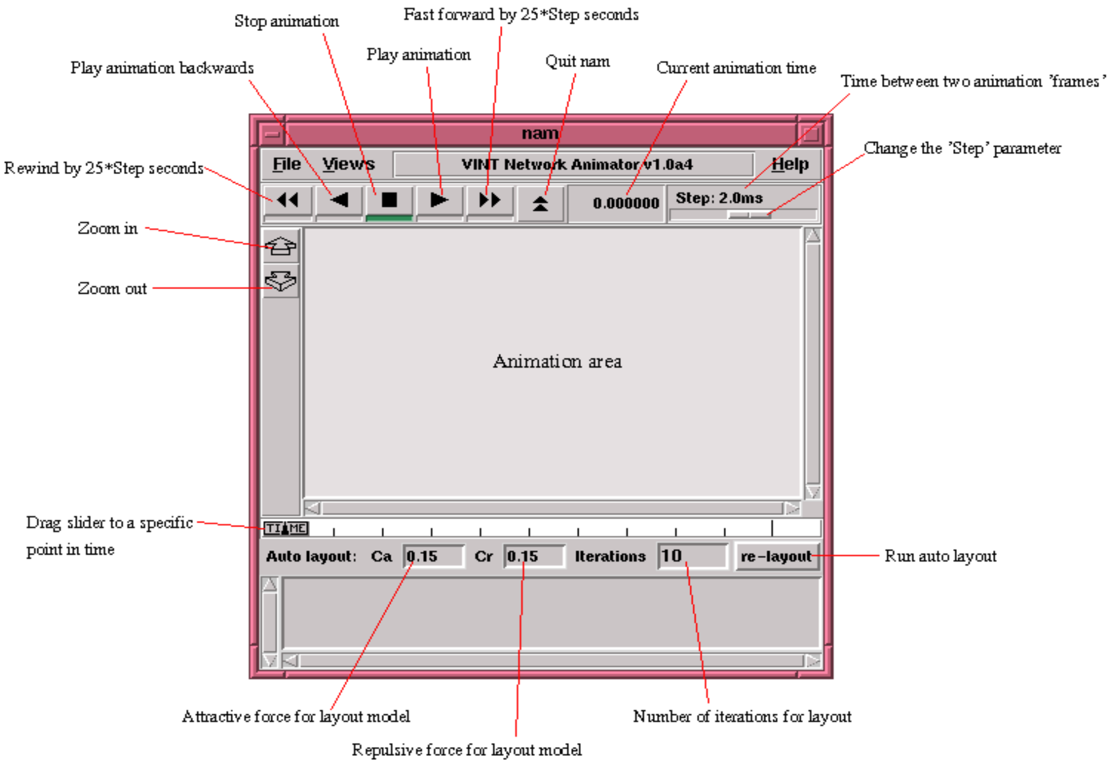
1.6 NS-2 features

NS-2 implements the following features

1. Router queue Management Techniques DropTail, RED, CBQ,
2. Multicasting
3. Simulation of wireless networks
    - Developed by Sun Microsystems + UC Berkeley (Daedalus Project)
    - Terrestrial (cellular, adhoc, GPRS, WLAN, BLUETOOTH), satellite
    - IEEE 802.11 can be simulated, Mobile-IP, and adhoc protocols such as
    - DSR, TORA, DSDV and AODV.
4. Traffic Source Behaviour- www, CBR, VBR
5. Transport Agents- UDP/TCP
6. Routing
7. Packet flow
8. Network Topology
9. Applications- Telnet, FTP, Ping
10. Tracing Packets on all links/specific links

1.7 NAM (Network Animator)

NAM provides a visual interpretation of the network topology created. Below you can see a screenshot of a nam window where the most important functions are being explained.


Stop animation
Fast forward by 25*Step seconds
Play animation
Play animation backwards
Quit nam
Current animation time
Time between two animation 'frames'
Change the 'Step' parameter
Rewind by 25*Step seconds
Zoom in
Zoom out
nam
File
Views
VINT Network Animator v1.0a4
Help
0.000000
Step: 2.0ms
Animation area
Drag slider to a specific point in time
TIME
Auto layout:
Ca
0.15
Cr
0.15
Iterations
10
re-layout
Run auto layout
Attractive force for layout model
Repulsive force for layout model
Number of iterations for layout

Its features are as follows:

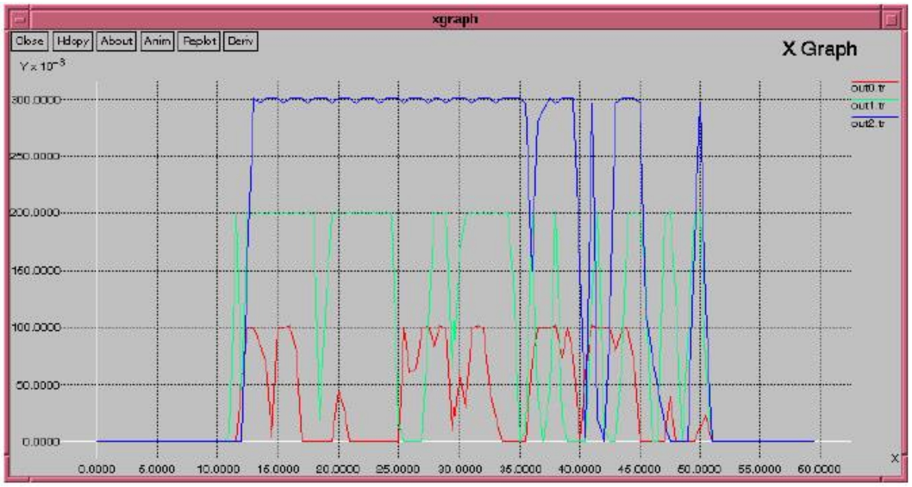
- Provides a visual interpretation of the network created
- Can be executed directly from a Tcl script
- Controls include play, stop ff, rw, pause, a display speed controller and a packet monitor facility.
- Presents information such as throughput, number packets on each link.
- Provides a drag and drop interface for creating topologies.

1.8 XGraph

XGraph is an X-Windows application that includes:

- Interactive plotting and graphing
- Animation and derivatives

To use XGraph in NS-2 the executable can be called within a TCL Script. This will then load a graph displaying the information visually displaying the information of the trace file produced from the simulation.



XGraph running comparing three trace files in a graph

1.9 TraceGraph

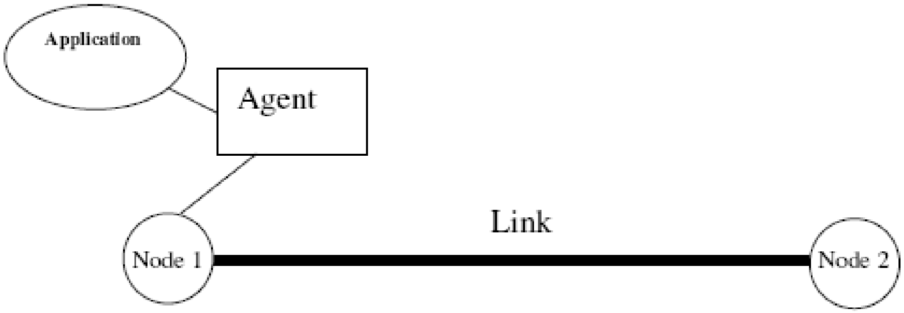
TraceGraph is a trace file analyser that runs under Windows, Linux and UNIX systems and requires Matlab 6.0 or higher.

TraceGraph supports the following trace file formats.

- Wired
- Satellite
- Wireless

2. OTcl Scripting with NS-2

In NS-2, the network is constructed using nodes which are connected using links. Events are scheduled to pass between nodes through the links. Nodes and links can have various properties associated with them. Agents can be associated with nodes and they are responsible for generating different packets (e.g. TCP agent or UDP agent). The traffic source is an application which is associated with a particular agent (e.g. ping application).



This diagram shows two nodes, a link, an agent and an application.

2.1 How to start ?

First of all, you need to create a simulator object. This is done with the command

```
     set ns [new Simulator]
```

Now we open a file for writing that is going to be used for the nam trace data.

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

The first line opens the file 'out.nam' for writing and gives it the file handle 'nf'. In the second line we tell the simulator object that we created above to write all simulation data that is going to be relevant for nam into this file. The next step is to add a 'finish' procedure that closes the trace file and starts nam.

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

The last line finally starts the simulation

```
      $ns run
```

2.2 Node creation and linking

The following two lines define the two nodes. (Note: You have to insert the code in this section **before** the line '$ns run', or even better, before the line '$ns at 5.0 "finish"').

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

A new node object is created with the command '$ns node'. The above code creates two nodes and assigns them to the handles 'n0' and 'n1'. The next line connects the two nodes.

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

This line tells the simulator object to connect the nodes n0 and n1 with a duplex link with the bandwidth 1Megabit, a delay of 10ms and a DropTail queue.

Now you can save your file and start the script with 'ns example1.tcl'. nam will be started automatically and you should see an output that resembles the picture below.

2.3 Sending data

The next step is to send some data from node n0 to node n1. In ns, data is always being sent from one 'agent' to another. So the next step is to create an agent object that sends data from node n0, and another agent object that receives the data on node n1.

```
#Create a UDP agent and attach it to node n0
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
```

These lines create a UDP agent and attach it to the node n0, then attach a CBR traffic generator to the UDP agent. CBR stands for 'constant bit rate'. Line 7 and 8 should be self-explaining. The packet Size is being set to 500 bytes and a packet will be sent every 0.005 seconds (i.e. 200 packets per second). The next lines create a Null agent which acts as traffic sink and attach it to node n1.

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

Now the two agents have to be connected with each other.

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

And now we have to tell the CBR agent when to send data and when to stop sending. It's probably best to put the following lines just before the line '$ns at 5.0 "finish"'.

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

Now you can save the file and start the simulation again. When you click on the 'play' button in the nam window, you will see that after 0.5 simulation seconds, node 0 starts sending data packets to node 1. You might want to slow nam down then with the 'Step' slider.

Add the following two lines to your CBR agent definitions.

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

The parameter 'fid_' stands for 'flow id'.

Now you can add the following piece of code to your Tcl script, preferably at the beginning after the simulator object has been created, since this is a part of the simulator setup.

```
        $ns color 1 Blue
        $ns color 2 Red
```

This code allows you to set different colors for each flow id.

You can add the following line to your code to monitor the queue for the link from n2 to n3.

```
        $ns duplex-link-op $n2 $n3 queuePos 0.5
```

You can see the packets in the queue now, and after a while you can even see how the packets are being dropped, though (at least on my system, I guess it might be different in later or earlier releases) only blue packets are being dropped. But you can't really expect too much 'fairness' from a simple Drop Tail queue. So let's try to improve the queuing by using a SFQ (stochastic fair queuing) queue for the link from n2 to n3. Change the link definition for the link between n2 and n3 to the following line.

```
$ns duplex-link $n3 $n2 1Mb 10ms SFQ
```

The queuing should be 'fair' now. The same amount of blue and red packets should be dropped.

Experiment No. 1

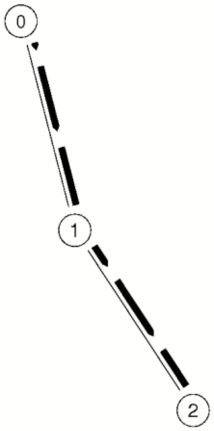
Problem Statement:

Simulate a 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.

```
#File Name: nsexp1.tcl
#Description: Simulating simple three nodes point-to-point network
##################################################################

set ns [new Simulator]

#Open a new file for NAMTRACE
set nf [open out.nam w]
$ns namtrace-all $nf

#Open a new file to log TRACE
set tf [open out.tr w]
$ns trace-all $tf

#Body of the finish procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        close $nf
        close $tf
        exec nam out.nam &
        exit 0
}

#Create Nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

#Create Links between Nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
```

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

#Set the queue limit - default is 50 packets
$ns queue-limit $n0 $n1 50
$ns queue-limit $n1 $n2 50

#Create Transport Agent
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set null0 [new Agent/Null]
$ns attach-agent $n2 $null0
$ns connect $udp0 $null0

#Create Application to generate traffic
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Start and Stop generating traffic
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

#Stop the simulation
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

Nam output:



Experiment No. 2

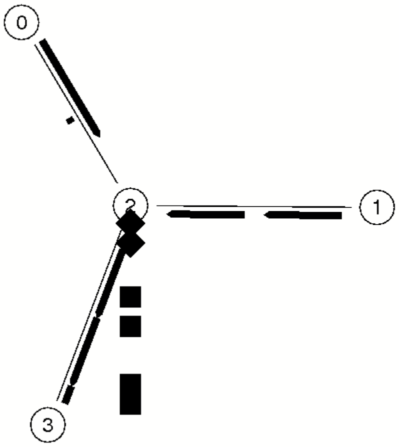
Problem Statement:

Simulate a four node point-to-point network, and connect the links as follows: n0-n2, n1-n2 and n2-n3. Apply TCP agent between n0-n3 and UDP n1-n3. Apply relevant applications over TCP and UDP agents changing the parameter and determine the number of packets by TCP/UDP.

```
#File Name: nsexp2.tcl

#Description:  Simulating four node point-to-point network with TCP and UDP agent
#################################################################################

set ns [new Simulator]

#Open a new file for NAMTRACE
set nf [open out.nam w]
$ns namtrace-all $nf

#Open a new file to log TRACE
set tf [open out.tr w]
$ns trace-all $tf

#Body of the 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        close $nf
        close $tf
        exec nam out.nam &
        exit 0
}

#Create Nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create Links between Nodes
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail

#Set the queue limit - default is 50 packets
$ns queue-limit $n0 $n2 50
$ns queue-limit $n1 $n2 50
$ns queue-limit $n2 $n3 50

#Create TCP Agent between node 0 and node 3
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
```

```
#Create FTP Application for TCP Agent
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

#Specify TCP packet size
Agent/TCP set packetSize_ 1000

#Create UDP Agent between node 1 and node 3
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
$ns connect $udp0 $null0

#Create CBR Application for UDP Agent
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Start and Stop FTP Traffic
$ns at 0.75 "$ftp0 start"
$ns at 4.75 "$ftp0 stop"

#Start and Stop CBR traffic
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

#Stop the simulation
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

NAM output:



Experiment No. 3

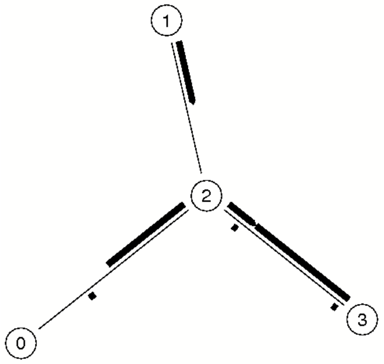
Problem Statement:

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

```
#File Name: nsexp3.tcl
#Description: Simulating different types of Internet traffic
############################################################

set ns [new Simulator]

#Open a new file for NAMTRACE
set nf [open out.nam w]
$ns namtrace-all $nf

#Open a new file to log TRACE
set tf [open out.tr w]
$ns trace-all $tf

#Body of the 'finish' procedure
proc finish {} {
        global ns nf tf
        $ns flush-trace
        close $nf
        close $tf
        exec nam out.nam &
        exit 0
}

#Create Nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create Links between Nodes
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail

#Set the queue limit - default is 50 packets
$ns queue-limit $n0 $n2 50
$ns queue-limit $n1 $n2 50
$ns queue-limit $n2 $n3 50

#Create TCP Agent between node 0 and node 3
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0

#Create FTP Application for TCP Agent
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

#Specify TCP packet size
```

```
Agent/TCP set packetSize_ 1000

#Create TCP Agent between node 1 and node 3
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1

#Create Telnet Application for TCP Agent
set telnet0 [new Application/Telnet]
$telnet0 set interval_ 0.005
$telnet0 attach-agent $tcp1

#Start and Stop FTP Traffic
$ns at 0.75 "$ftp0 start"
$ns at 4.75 "$ftp0 stop"

#Start and Stop Telnet traffic
$ns at 0.5 "$telnet0 start"
$ns at 4.5 "$telnet0 stop"

#Stop the simulation
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

NAM output



Experiment No. 4

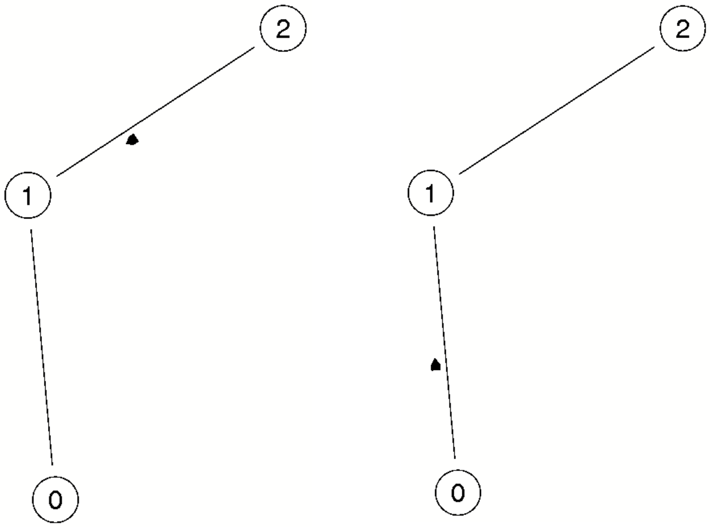
Problem Statement:

Simulate the transmission of ping messaged over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

```
#File Name: nsexp4.tcl

#Description: transmission of ping messaged over a network topology
###################################################################

#Create a simulator object
set ns [new Simulator]

#Open a trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
}

#Create three nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

#Connect the nodes with two links
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail

#Define a 'recv' function for the class 'Agent/Ping'
Agent/Ping instproc recv {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 the nodes n0 and n2
set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0

set p1 [new Agent/Ping]
$ns attach-agent $n2 $p1

#Connect the two agents
$ns connect $p0 $p1

#Schedule events
$ns at 0.2 "$p0 send"
$ns at 0.4 "$p1 send"
$ns at 0.6 "$p0 send"
```

```
$ns at 0.6 "$p1 send"
$ns at 1.0 "finish"

#Run the simulation
$ns run
```

NAM output:



Part B Experiments

Experiment No. 1

Problem Statement:

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

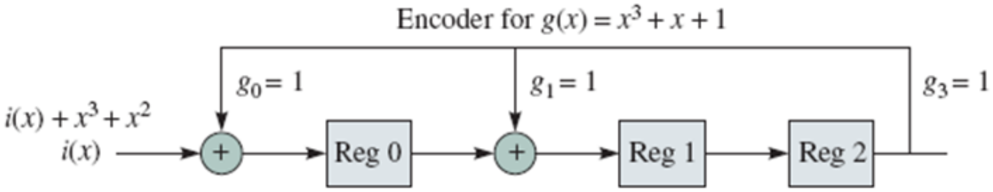
Theory:

It does error checking via polynomial division. In general, a bit string can be represented using a polynomial. Ex: 10010101110 is same as $X^{10} + X^7 + X^5 + X^3 + X^2 + X$. All computations are done in modulo 2.

Algorithm:

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

The algorithm can be implemented using a feedback shift-register as shown below.



Clock	Input	Reg 0	Reg 1	Reg 2
0	-	0	0	0
1	$1 = i_3$	1	0	0
2	$1 = i_2$	1	1	0
3	$0 = i_1$	0	1	1
4	$0 = i_0$	1	1	1
5	0	1	0	1
6	0	1	0	0
7	0	**0**	**1**	**0**
	Check bits:	$\boldsymbol{r_0 = 0}$	$\boldsymbol{r_1 = 1}$	$\boldsymbol{r_2 = 0}$

$\rightarrow \boldsymbol{r(x) = x}$

Experiment No. 2

Problem Statement:

Write a program for Distance Vector Algorithm to find suitable path for transmission.

Theory:

The distance vector routing algorithm is sometimes called by other names, including the distributed Bellman-Ford routing algorithm and the Ford-Fulkerson algorithm, after the researchers who developed it. In distance vector routing, each router maintains a routing table indexed by, and containing one entry for, each router in subnet. This entry contains two parts: the preferred out going line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

The router is assumed to know the “distance” to each of its neighbor. If the metric is hops, the Distance is just one hop. If the metric is queue length, the router simply examines each queue. If the metric is delay, the router can measure it directly with special ECHO packets that the receiver just time stamps and sends back as fast as possible.

Algorithm:

Experiment No. 3

Problem Statement:

Write a program for Hamming Code generation for error detection and correction

Theory:

Hamming codes (**Richard Hamming,** 1950) are used for detecting and correcting single bit errors in transmitted data. This requires that 3 parity bits (check bits) be transmitted with every 4 data bits. The algorithm is called A(7, 4) code, because it requires seven bits to encode 4 bits of data.

Algorithm (encoding)

1. Index (k+ r) bits starting from 1. E.g . bit 1, 2, 3, 4, 5, etc.
2. Write the bit position numbers in binary. i.e. 1, 10, 11, 100, 101, etc.
3. All bit positions that are powers of two are parity bits.
4. All other bit positions are data bits.
5. Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.

Bit position		01	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1010	1111	10000	10001	10010	10011	10100	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	X		X		X		X		X		X		X		X		X		X		• • •
	p2		X	X			X	X			X	X			X	X			X	X		
	p4				X	X	X	X					X	X	X	X					X	
	p8								X	X	X	X	X	X	X	X						
	p16																X	X	X	X	X	

Algorithm (decoding)

1. To check for errors, check all of the parity bits. If all parity bits are correct, there is no error.
2. Otherwise, the sum of the positions of the erroneous parity bits identifies the erroneous bit.

3. If only one parity bit indicates an error, the parity bit itself is in error. Erroneous parity bit means no. of one 1's is odd. Correct parity bit means no. of one 1's is even.
4. Extract the data bits if there is no error.

Experiment No. 4

Problem Statement:

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

Theory:

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977. 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.

Algorithm (*computing public key and the private key*)

1. Choose two large prime numbers, p and q
2. Compute n = p x q and z = (p - 1) x (q - 1).
3. Choose a number that is relatively prime to **z** and call it **e**

   i.e. gcd (z, e) =1

4. Find **d** such that e x d = 1 mod z.

   Public key = {e, n} and private key = {d, n}.

Algorithm (*encryption and decryption*)

Let P be plaintext (an Integer) to be encrypted, $0 \leq P \leq n$

1. To encrypt compute $C = P^e$ (mod n)
2. To decrypt C, compute $P = C^d$ (mod n).

RSA is based on the following key property:

$$\mathbf{P^{de} \text{ (mod n)} = P \text{ (mod n)}}$$

Modular arithmetic involving large numbers can be simplified by using the following property.

$$\mathbf{(ab) \text{ mod } n = ((a \text{ mod } n)(b \text{ mod } n)) \text{ mod } n}$$

Experiment No. 5

Problem Statement:

Write a program for congestion control using Leaky bucket algorithm.

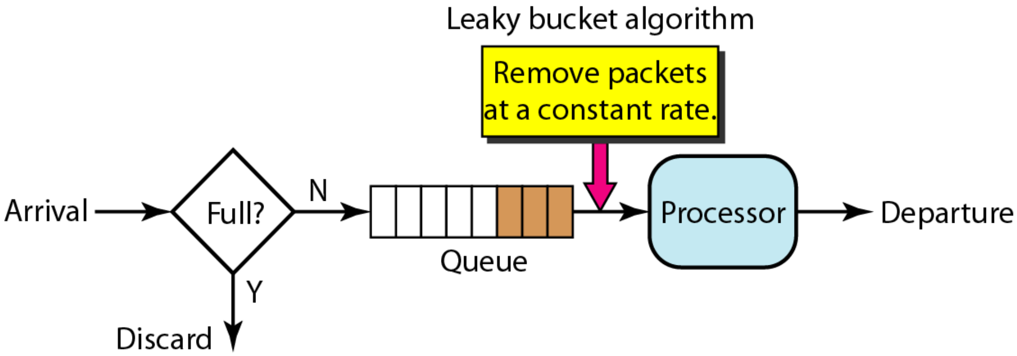
Theory:

Leaky bucket (proposed by Jonathan S. Turner, 1986) is a traffic shaping algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded.

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

Algorithm

1. The leaky bucket consists of a finite queue.
2. When a packet arrives, if there is room on the queue it is appended to the queue; otherwise, it is discarded.
3. At every clock tick, one packet is transmitted



Experiment No. 6

Problem Statement:

Write a program for frame sorting technique used in buffers.

Theory:

Algorithm

1. Read the message from Key board
2. Divide the message based on the size of the frame and assign sequence number
3. Shuffle the frames
4. Sort the received Frames
5. Print/Display the Result.