**EXPERIMENT NO. 2**

1. **Introduction to tcl programming and implementation of two node network using NS (Network Simulator)-2**

**Aim:** To introduce tcl (tool command language) programming.

**Requirements**

1. Ubuntu operating system
2. Network Simulator NS2 (preferably version 2.35)

**List of Programs to be conducted-**

1. Print ‘Hello World’ using tcl programming.
2. Display age (any data) using variables using tcl programming.
3. Input data from the end user using tcl programming.
4. To create a simple two node network using tcl programming.

**Introduction**

The network simulator is a discrete event packet level simulator. It covers a very large number of applications of different kinds of protocols of different network types consisting of different network elements and traffic models. Network simulator includes a package of tools that simulates behavior of networks such as creating network topologies, log events that happen under any load, analyze the events and understand the network. It is mainly based on two languages, C++ and O-Tcl (object oriented extension of Tcl).

NS needs certain supportive packages such as ‘Nam (network animator)’. Nam is a Tcl/TK based animation tool for viewing network simulation traces and real world packet traces. It supports topology layout, packet level animation, and various data inspection tools. NS 2 and Nam have evolved substantially over the past few years. Currently, they are being developed as an open source project hosted at Sourceforge.

Common Procedure for Tcl programming,

1. Open the editor.
2. Write the program.
3. Save the program.
4. Open the terminal.
5. Run/Execute the program using the ‘ns’ command.

The detailed procedure to create a simple two node network using tcl is given below. Creating node includes the following steps-

1. Starting a new simulator.
2. Creating output files.
3. Terminating the program.
4. Starting the simulation.
5. Defining nodes and links.
6. Attaching agents and applications.
7. Scheduling events.
8. Viewing the output in nam.
9. Tracing the output.

**Algorithm**

1. Start a new simulator.
2. Create the output trace file (.tr) and nam file (.nam)
3. Terminate the program.
4. End the program by calling ‘finish’ proc.
5. Start simulation
   1. Define nodes, links query and topology.
   2. Set agents and applications.
   3. Initiate FTP over TCP/UDP.
   4. Scheduling
   5. Run the simulation.
6. Check the out files.

**Implementation**

1. Open file using > gedit *expname.tcl* in the root directory (students can give other name to their files)
2. Write the program.
3. Run the program using ns *expname.tcl*
4. The output is in form of .nam and .tr file which are generated in the same directory

**Observations:** Take appropriate snapshots and write observations below each snapshot with a pen.

**Post Experimental Exercise:**

Note the observations and frame the appropriate conclusions.

1. Change the orientation of the nodes.
2. Change the bandwidth parameter and check the effect in output.
3. Change the delay parameter and check the output.

**Conclusion:**

**EXPERIMENT NO. 2**

1. **Implementation of four node network topology using NS-2**

**Aim:** To create a network consisting of four nodes in order to understand more features of Tcl programming in NS-2. This includes-

1. Formation of suitable topology
2. Marking flows and adding colors
3. Defining packet size, packet interval
4. Setting queue size of the link
5. Monitoring a queue

**Requirements**

Ubuntu operating system

NS2

**Procedure:** This program deals with data transfer between different nodes using a fixed topology. The steps to be followed are listed

1. Start a new simulator
2. Create the output files
3. Write the finish procedure
4. Create the required topology by,
   1. Defining nodes and the links
   2. Creating, attaching and connecting transport layer agents
   3. Creating and attaching application layer agent
   4. Scheduling the events
   5. Starting the simulation
5. Run the simulation
6. Observe the animated output in the ‘nam’ window.
7. Check the trace file

**Formation of suitable topology** ([Ref: https://www.isi.edu/nsnam/ns/tutorial/](about:blank))

When we run the ns-2 program in nam (network animator), it gives random positions to the nodes. We can give suitable initial positions to the nodes and can form a suitable topology.

    #Give position to the nodes in nam  
       
    $ns duplex-link-op $n0 $n2 orient-right-down  
    $ns duplex-link-op $n1 $n2 orient-right-up  
    $ns simplex-link-op $n2 $n3 orient-right  
    $ns simplex-link-op $n3 $n2 orient-left  
    $ns duplex-link-op $n3 $n4 orient-right-up  
    $ns duplex-link-op $n3 $n5 orient-right-down

**Marking flows and adding colors**

When more than two nodes are present, it becomes necessary to associate different class to different flows and then assign a single color to each class to easily follow the different flows in nam.

| $udp0 set class\_ 1  $udp1 set class\_ 2 |
| --- |

The parameter 'fid\_' stands for 'flow id'.

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.

**Defining packet size, packet interval**

We can set packet size by using the command

$cbr set packetSize\_(packetsize)

After defining the UDP source and UDP agent, instead of defining the rate we can define the time interval between the transmission of packets using the command

$cbr0 set interval\_ 0.005

Example of setting packet size and packet interval

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. The packetSize is being set to 500 bytes and a packet will be sent every 0.005 seconds (i.e. 200 packets per second). This results in a bandwidth of 0.8 megabits per second for the link between the two nodes. If the link bandwidth selected is less than 0.8 megabits, some packets are being discarded.

**Setting queue size of the link**

In NS an output queue of a node is implemented as a part of a link whose input is that node to handle the overflow at the queue. But if the buffer capacity of the output queue is exceeded then the last packet arrived is dropped and here we will use a 'DropTail' option. Many other options such as RED (Random Early Discard) mechanism, FQ (Fair Queuing), DRR (Deficit Round Robin), SFQ (Stochastic Fair Queuing) are available.

So now we will define the buffer capacity of the queue related to the above link

    #Set queue size of the link  
    $ns queue-limit $n0 $n2 20

**Monitoring a queue**

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.

**Observations:**

Take appropriate screenshots and write observations below each. Students are expected to take minimum of four snapshots for different periods of simulations.

**Post Experimental Exercise**

1. What is a ‘queue’ in computer networks? Explain different queuing mechanisms in brief. (Ref: [*https://sites.google.com/a/seecs.edu.pk/network-technologies-tcp-ip-suite/home/performance-analysis-of-impact-of-various-queuing-mechanisms-on-mpeg-traffic/working-mechanism-of-fq-red-sfq-drr-and-drop-tail-queues*](https://sites.google.com/a/seecs.edu.pk/network-technologies-tcp-ip-suite/home/performance-analysis-of-impact-of-various-queuing-mechanisms-on-mpeg-traffic/working-mechanism-of-fq-red-sfq-drr-and-drop-tail-queues)*)*
2. In your simulation, change the color of the packets moving from node 1 to 2 and 2 to 3 and observe the output. (take screenshot and attach as the output)

**Conclusion:**