Use command ns to check if the installation is done or not.

If Yes it give % symbol

#Create a simulator object

set ns [new Simulator]

#Define different colors for data flows (for NAM)

$ns color 1 Blue

$ns color 2 Red

#Open the NAM 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 the NAM trace file

close $nf

#Execute NAM on the trace file

exec nam out.nam &

exit 0

}

#Create four nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

#Create links between the nodes

$ns duplex-link $n0 $n2 2Mb 10ms DropTail

$ns duplex-link $n1 $n2 2Mb 10ms DropTail

$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

#Set Queue Size of link (n2-n3) to 10

$ns queue-limit $n2 $n3 10

#Give node position (for NAM)

$ns duplex-link-op $n0 $n2 orient right-down

$ns duplex-link-op $n1 $n2 orient right-up

$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for link (n2-n3). (for NAM)

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

#Setup a TCP connection

set tcp [new Agent/TCP]

$tcp set class\_ 2

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]

$ns attach-agent $n3 $sink

$ns connect $tcp $sink

$tcp set fid\_ 1

#Setup a FTP over TCP connection

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ftp set type\_ FTP

#Setup a UDP connection

set udp [new Agent/UDP]

$ns attach-agent $n1 $udp

set null [new Agent/Null]

$ns attach-agent $n3 $null

$ns connect $udp $null

$udp set fid\_ 2

#Setup a CBR over UDP connection

set cbr [new Application/Traffic/CBR]

$cbr attach-agent $udp

$cbr set type\_ CBR

$cbr set packet\_size\_ 1000

$cbr set rate\_ 1mb

$cbr set random\_ false

#Schedule events for the CBR and FTP agents

$ns at 0.1 "$cbr start"

$ns at 1.0 "$ftp start"

$ns at 4.0 "$ftp stop"

$ns at 4.5 "$cbr stop"

#Detach tcp and sink agents (not really necessary)

$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

#Call the finish procedure after 5 seconds of simulation time

$ns at 5.0 "finish"

#Print CBR packet size and interval

puts "CBR packet size = [$cbr set packet\_size\_]"

puts "CBR interval = [$cbr set interval\_]"

#Run the simulation

$ns run

To execute use filename.tcl

The following is the explanation of the script above. In general, an NS script starts with making a Simulator object instance.

* set *ns* [new Simulator]: generates an NS simulator object instance, and assigns it to variable *ns* (italics is used for variables and values in this section). What this line does is the following:  
  + Initialize the packet format (ignore this for now)
  + Create a scheduler (default is calendar scheduler)
  + Select the default address format (ignore this for now)

The "Simulator" object has member functions that do the following:

* + Create compound objects such as nodes and links (described later)
  + Connect network component objects created (ex. attach-agent)
  + Set network component parameters (mostly for compound objects)
  + Create connections between agents (ex. make connection between a "tcp" and "sink")
  + Specify NAM display options
  + Etc.

Most of member functions are for simulation setup (referred to as plumbing functions in the Overview section) and scheduling, however some of them are for the NAM display. The "Simulator" object member function implementations are located in the "ns-2/tcl/lib/ns-lib.tcl" file.

* *$ns* color *fid color*: is to set color of the packets for a flow specified by the flow id (fid). This member function of "Simulator" object is for the NAM display, and has no effect on the actual simulation.
* *$ns* namtrace-all *file-descriptor*: This member function tells the simulator to record simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command *$ns*flush-trace. Similarly, the member function trace-all is for recording the simulation trace in a general format.
* proc *finish* {}: is called after this simulation is over by the command *$ns* at 5.0 "*finish*". In this function, post-simulation processes are specified.
* set *n0* [*$ns* node]: The member function node creates a node. A node in NS is compound object made of address and port classifiers (described in a later section). Users can create a node by separately creating an address and a port classifier objects and connecting them together. However, this member function of Simulator object makes the job easier. To see how a node is created, look at the files: "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-node.tcl".
* *$ns* duplex-link *node1 node2 bandwidth delay queue-type*: creates two simplex links of specified bandwidth and delay, and connects the two specified nodes. In NS, the output queue of a node is implemented as a part of a link, therefore users should specify the queue-type when creating links. In the above simulation script, DropTail queue is used. If the reader wants to use a RED queue, simply replace the word DropTail with RED. The NS implementation of a link is shown in a later section. Like a node, a link is a compound object, and users can create its sub-objects and connect them and the nodes. Link source codes can be found in "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl" files. One thing to note is that you can insert error modules in a link component to simulate a lossy link (actually users can make and insert any network objects). Refer to the NS documentation to find out how to do this.
* *$ns* queue-limit *node1 node2 number*: This line sets the queue limit of the two simplex links that connect node1 and node2 to the number specified. At this point, the authors do not know how many of these kinds of member functions of Simulator objects are available and what they are. Please take a look at "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl", or NS documentation for more information.
* *$ns* duplex-link-op *node1 node2 ...*: The next couple of lines are used for the NAM display. To see the effects of these lines, users can comment these lines out and try the simulation.

Now that the basic network setup is done, the next thing to do is to setup traffic agents such as TCP and UDP, traffic sources such as FTP and CBR, and attach them to nodes and agents respectively.

* set *tcp* [new *Agent/TCP*]: This line shows how to create a TCP agent. But in general, users can create any agent or traffic sources in this way. Agents and traffic sources are in fact basic objects (not compound objects), mostly implemented in C++ and linked to OTcl. Therefore, there are no specific Simulator object member functions that create these object instances. To create agents or traffic sources, a user should know the class names these objects (Agent/TCP, Agnet/TCPSink, Application/FTP and so on). This information can be found in the NS documentation or partly in this documentation. But one shortcut is to look at the "ns-2/tcl/libs/ns-default.tcl" file. This file contains the default configurable parameter value settings for available network objects. Therefore, it works as a good indicator of what kind of network objects are available in NS and what are the configurable parameters.
* *$ns* attach-agent *node agent*: The attach-agent member function attaches an agent object created to a node object. Actually, what this function does is call the attach member function of specified node, which attaches the given agent to itself. Therefore, a user can do the same thing by, for example, $n0 attach $tcp. Similarly, each agent object has a member function attach-agent that attaches a traffic source object to itself.
* *$ns* connect *agent1 agent2*: After two agents that will communicate with each other are created, the next thing is to establish a logical network connection between them. This line establishes a network connection by setting the destination address to each other’s' network and port address pair.

Assuming that all the network configuration is done, the next thing to do is write a simulation scenario (i.e. simulation scheduling). The Simulator object has many scheduling member functions. However, the one that is mostly used is the following:

* *$ns* at *time "string"*: This member function of a Simulator object makes the scheduler (scheduler\_ is the variable that points the scheduler object created by [new Scheduler] command at the beginning of the script) to schedule the execution of the specified string at given simulation time. For example, *$ns* at *0.1 "$cbr start"* will make the scheduler call a start member function of the CBR traffic source object, which starts the CBR to transmit data. In NS, usually a traffic source does not transmit actual data, but it notifies the underlying agent that it has some amount of data to transmit, and the agent, just knowing how much of the data to transfer, creates packets and sends them.

After all network configuration, scheduling and post-simulation procedure specifications are done, the only thing left is to run the simulation. This is done by *$ns* run.

Introduction

The network simulator is discrete event packet level simulator. The network simulator covers a very large number of application of different kind of protocols of different network types consisting of different network elements and traffic models.Network simulator is 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. Well the main aim of our first experiment is to learn how to use network simulator and to get acquainted with the simulated objects and understand the operations of network simulation and we also need to analyze the behavior of the simulation object using network simulation.

Platform required to run network simulator

Unix and Unix like systems

Linux (Use Fedora or Ubuntu versions)

Free BSD

SunOS/Solaris

Windows 95/98/NT/2000/XP

Backend Environment of Network Simulator

Network Simulator is mainly based on two languages. They are C++ and OTcl. OTcl is the object oriented version of Tool Command language.The network simulator is a bank of different network and protocol objects. C++ helps in the following way:

It helps to increase the efficiency of simulation.

Its is used to provide details of the protocols and their operation.

It is used to reduce packet and event processing time.

OTcl helps in the following way:

With the help of OTcl we can describe different network topologies

It helps us to specify the protocols and their applications

It allows fast development

Tcl is compatible with many platforms and it is flexible for integration

Tcl is very easy to use and it is available in free

Basics of Tcl Programming (w.r.t. ns2)

Before we get into the program we should consider the following things:

Initialization and termination aspects of network simulator.

Defining the network nodes,links,queues and topology as well.

Defining the agents and their applications

Network Animator(NAM)

Tracing

Initialization

To start a new simulator we write

set ns [new Simulator]

From the above command we get that a variable ns is being initialized by using the set command. Here the code [new Simulator] is a instantiation of the class Simulator which uses the reserved word 'new'. So we can call all the methods present inside the class simulator by using the variable ns.

Creating the output files

1 #To create the trace files we write

2

3 set tracefile1 [open out.tr w]

4 $ns trace-all $tracefile1

5

6 #To create the nam files we write

7

8 set namfile1 [open out.nam w]

9 $ns namtrace-all $namfile

In the above we create a output trace file out.tr and a nam visualization file out.nam. But in the Tcl script they are not called by their names declared,while they are called by the pointers initialized for them such as tracefile1 and namfile1 respectively.The line which starts with '#' are commented.The next line opens the file 'out.tr' which is used for writing is declared 'w'.The next line uses a simulator method trace-all by which we will trace all the events in a particular format.

The termination program is done by using a 'finish' procedure

01 # Defining the 'finish' procedure'

02

03 proc finish {} {

04 global ns tracefile1 namfile1

05 $ns flush-trace

06 close $tracefile

07 close $namfile

08 exec nam out.nam &

09 exit 0

10 }

In the above the word 'proc' is used to declare a procedure called 'finish'.The word 'global' is used to tell what variables are being used outside the procedure.

'flush-trace' is a simulator method that dumps the traces on the respective files.the command 'close' is used to close the trace files and the command 'exec' is used to execute the nam visualization.The command 'exit' closes the application and returns 0 as zero(0) is default for clean exit.

In ns we end the program by calling the 'finish' procedure

1 #end the program

2 $ns at 125.0 "finish"

Thus the entire operation ends at 125 seconds. To begin the simulation we will use the command

1 #start the the simulation process

2 $ns run

Defining nodes,links,queues and topology

Way to create a node:

view source

print?

1 set n0 [ns node]

In the above we created a node that is pointed by a variable n0.While referring the node in the script we use $n0. Similarly we create another node n2.Now we will set a link between the two nodes.

1 $ns duplex-link $n0 $n2 10Mb 10ms DropTail

So we are creating a bi-directional link between n0 and n2 with a capacity of 10Mb/sec and a propagation delay of 10ms.

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 'Drop Tail' 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

1 #Set queue size of the link

2 $ns queue-limit $n0 $n2 20

so, if we summarize the above three things we get

01 #create nodes

02

03 set n0 [$ns node]

04 set n1 [$ns node]

05 set n2 [$ns node]

06 set n3 [$ns node]

07 set n4 [$ns node]

08 set n5 [$ns node]

09

10 #create links between the nodes

11

12 $ns duplex-link $n0 $n2 10Mb 10ms DropTail

13 $ns duplex-link $n1 $n2 10Mb 10ms DropTail

14 $ns simplex-link $n2 $n3 0.3Mb 100ms DropTail

15 $ns simplex-link $n3 $n2 0.3Mb 100ms DropTail

16 $ns duplex-link $n0 $n2 0.5Mb 40ms DropTail

17 $ns duplex-link $n0 $n2 0.5Mb 40ms DropTail

18

19 #set queue-size of the link (n2-n3) to 20

20 $ns queue-limit $n2 $n3 20

Agents and applications

TCP

TCP is a dynamic reliable congestion protocol which is used to provide reliable transport of packets from one host to another host by sending acknowledgements on proper transfer or loss of packets.Thus TCP requires bi-directional links in order for acknowledgements to return to the source.

Now we will show how to set up tcp connection between two nodes

1 #setting a tcp connection

2

3 set tcp [new Agent/TCP]

4 $ns attach-agent $n0 $tcp

5 set sink [new Agent/TCPSink]

6 $ns attach-agent $n4 $sink

7 $ns connect $tcp $sink

8 $tcp set fid\_1

9 $tcp set packetSize\_552

The command 'set tcp [new Agent/TCP]' gives a pointer called 'tcp' which indicates the tcp agent which is a object of ns.Then the command '$ns attach-agent $n0 $tcp' defines the source node of tcp connection. Next the command 'set sink [new Agent/TCPSink]' defines the destination of tcp by a pointer called sink. The next command '$ns attach-agent $n4 $sink' defines the destination node as n4.Next, the command '$ns connect $tcp $sink' makes the TCP connection between the source and the destination.i.e n0 and n4.When we have several flows such as TCP, UDP etc in a network. So, to identify these flows we mark these flows by using the command '$tcp set fid\_1'. In the last line we set the packet size of tcp as 552 while the default packet size of tcp is 1000.

FTP over TCP

File Transfer Protocol(FTP) is a standard mechanism provided by the Internet for transferring files from one host to another. Well this is the most common task expected from a networking or a inter networking . FTP differs from other client server applications in that it establishes between the client and the server. One connection is used for data transfer and other one is used for providing control information. FTP uses the services of the TCP. It needs two connections. The well Known port 21 is used for control connections and the other port 20 is used for data transfer.

Well here we will learn in how to run a FTP connection over a TCP

1 #Initiating FTP over TCP

2

3 set ftp [new Application/FTP]

4 $ftp attach-agent $tcp

In above,the command 'set ftp [new Application/FTP]' gives a pointer called 'ftp' which indicates the FTP application.Next, we attach the ftp application with tcp agent as FTP uses the services of TCP.

UDP

The User datagram Protocol is one of the main protocols of the Internet protocol suite.UDP helps the host to send send messages in the form of datagrams to another host which is present in a Internet protocol network without any kind of requirement for channel transmission setup. UDP provides a unreliable service and the datagrams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.

Now we will learn how to create a UDP connection in network simulator.

1 # setup a UDP connection

2 set udp [new Agent/UDP]

3 $ns attach-agent $n1 $udp

4 $set null [new Agent/Null]

5 $ns attach-agent $n5 $null

6 $ns connect $udp $null

7 $udp set fid\_2

Similarly,the command 'set udp [new Agent/UDP]' gives a pointer called 'udp' which indicates the udp agent which is a object of ns.Then the command '$ns attach-agent $n1 $udp' defines the source node of udp connection. Next the command 'set null [new Agent/Null]' defines the destination of udp by a pointer called null. The next command '$ns attach-agent $n5 $null' defines the destination node as n5.Next, the command '$ns connect $udp $null' makes the UDP connection between the source and the destination.i.e n1 and n5.When we have several flows such as TCP,UDP etc in a network. So, to identify these flows we mark these flows by using the command '$udp set fid\_2

Constant Bit Rate(CBR)

Constant Bit Rate (CBR) is a term used in telecommunications, relating to the quality of service.When referring to codecs, constant bit rate encoding means that the rate at which a codec's output data should be consumed is constant. CBR is useful for streaming multimedia content on limited capacity channels since it is the maximum bit rate that matters, not the average, so CBR would be used to take advantage of all of the capacity. CBR would not be the optimal choice for storage as it would not allocate enough data for complex sections (resulting in degraded quality) while wasting data on simple sections.

CBR over UDP Connection

1 #setup cbr over udp

2

3 set cbr [new Application/Traffic/CBR]

4 $cbr attach-agent $udp

5 $cbr set packetSize\_1000

6 $cbr set rate\_0.01Mb

7 $cbr set random \_false

In the above we define a CBR connection over a UDP one. Well we have already defined the UDP source and UDP agent as same as TCP. Instead of defining the rate we define the time interval between the transmission of packets in the command '$cbr set rate\_0.01Mb'. Next, with the help of the command '$cbr set random \_false' we can set random noise in cbr traffic.we can keep the noise by setting it to 'false' or we can set the noise on by the command '$cbr set random \_1'. We can set by packet size by using the command '$cbr set packetSize\_(packetsize).We can set the packet size up to sum value in bytes.

Scheduling Events

In ns the tcl script defines how to schedule the events or in other words at what time which event will occur and stop. This can be done using the command

$ns at .

So here in our program we will schedule the ftp and cbr.

1 # scheduling the events

2

3 $ns at 0.1 "cbr start"

4 $ns at 1.0 "ftp start"

5 $ns at 124.0 "ftp stop"

6 $ns at 124.5 "cbr stop"

Network Animator(NAM)

When we will run the above program in ns then we can can visualize the network in the NAM. But instead of giving random positions to the nodes, we can give suitable initial positions to the nodes and can form a suitable topology. So, in our program we can give positions to the nodes in NAM in the following way

1 #Give position to the nodes in NAM

2

3 $ns duplex-link-op $n0 $n2 orient-right-down

4 $ns duplex-link-op $n1 $n2 orient-right-up

5 $ns simplex-link-op $n2 $n3 orient-right

6 $ns simplex-link-op $n3 $n2 orient-left

7 $ns duplex-link-op $n3 $n4 orient-right-up

8 $ns duplex-link-op $n3 $n5 orient-right-down

We can also define the color of cbr and tcp packets for identification in NAM.For this we use the following command

1 #Marking the flows

2 $ns color1 Blue

3 $ns color2 Red

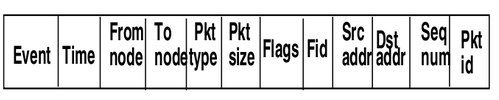
To view the network animator we need to type the command: nam

Tracing

Tracing Objects

NS simulation can produce visualization trace as well as ASCII file corresponding to the events that are registered at the network. While tracing ns inserts four objects: EnqT,DeqT,RecvT & DrpT. EnqT registers information regarding the arrival of packet and is queued at the input queue of the link. When overflow of a packet occurs, then the information of thye dropped packet is registered in DrpT.DeqT holds the information abut the packet that is dequeued instantly.RecvT hold the information about the packet that has been received instantly.

Structure of Trace files



The first field is event.It gives you four possible symbols '+' '-' 'r' 'd'.These four symbols correspond respectively to enqueued, dequeued, received and dropped.

The second field gives the time at which the event occurs

The third field gives you the input node of the link at which the event occurs

The fourth field gives you the the output node at which the event occurs

The fifth field shows the information about the packet type.i.e whether the packet is UDP or TCP

The sixth field gives the packet size

The seventh field give information about some flags

The eight field is the flow id(fid) for IPv6 that a user can set for each flow in a tcl script.It is also used for specifying the color of flow in NAM display

The ninth field is the source address

The tenth field is the destination address

The eleventh field is the network layer protocol's packet sequence number

The last field shows the unique id of packet

Following are trace of two events:

r 1.84471 2 1 cbr 210 ------- 1 3.0 1.0 195 600

r 1.84566 2 0 ack 40 ------- 2 3.2 0.1 82 602

The trace file can be viewed with the cat command:

cat out.tr