Simple Simulation Example

This section shows a simple NS simulation script and explains what each line does. Example 3 is an OTcl script that creates the simple network configuration and runs the simulation scenario in Figure 4. To run this simulation, download "ns-simple.tcl" and type "ns ns-simple.tcl" at your shell prompt.

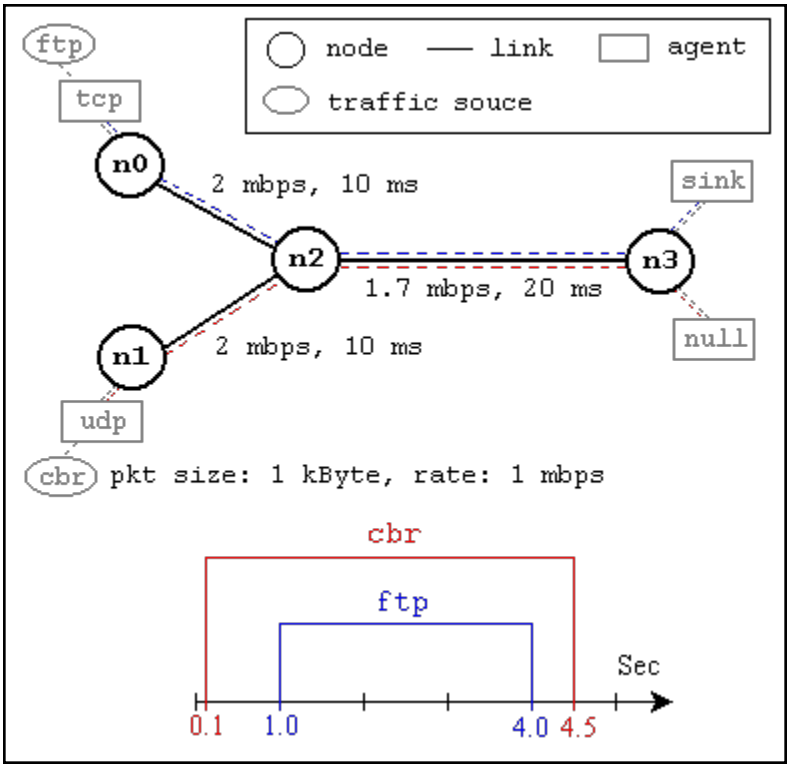

Figure 4. A Simple Network Topology and Simulation Scenario

This network consists of 4 nodes (n0, n1, n2, n3) as shown in above figure. The duplex links between n0 and n2, and n1 and n2 have 2 Mbps of bandwidth and 10 ms of delay. The duplex link between n2 and n3 has 1.7 Mbps of bandwidth and 20 ms of delay. Each node uses a DropTail queue, of which the maximum size is 10. A "tcp" agent is attached to n0, and a connection is established to a tcp "sink" agent attached to n3. As default, the maximum size of a packet that a "tcp" agent can generate is 1KByte. A tcp "sink" agent generates and sends ACK packets to the sender (tcp agent) and frees the received packets. A "udp" agent that is attached to n1 is connected to a "null" agent attached to n3. A "null" agent just frees the packets received. A "ftp" and a "cbr" traffic generator are attached to "tcp" and "udp" agents respectively, and the "cbr" is configured to

generate 1 KByte packets at the rate of 1 Mbps. The "cbr" is set to start at 0.1 sec and stop at 4.5 sec, and "ftp" is set to start at 1.0 sec and stop at 4.0 sec.

```
#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
```

Example 3. A Simple NS Simulation Script

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 others' 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.