Network Simulator 2 (NS2)

NS2 is a discrete event simulator targeted at networking research. NS2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. This simulator is primarily Unix based, and the NS commands can either be entered via Unix command prompt or by running a scripting language file, but scripting language is the preferred way. NS2 uses Tcl (pronounced as 'tickle') as its scripting language, which is edited using a text editor and in this example we will use vi.

Intro to vi

Vi is a text editor that is available in all Unix systems, other editors such pico and emacs can also be used instead. Before you start for the first time, you must note that vi is modal editor. A mode is like an environment. Different modes in vi interpret the same key differently. For example, if you’re in insert mode, typing a adds an a to the text, whereas in command mode, typing a puts you in insert mode because a is the key abbreviation for the append command. If you get confused about what mode you’re in, press the Escape key on your keyboard. Pressing Escape always returns you to the command mode (and if you’re already in command mode, it simply beeps to remind you of that fact).

When you are in command mode, you can manage your document; this includes the capability to change text, rearrange it, and delete it. A summary of the basic vi commands are available in Appendix A. Insert mode is when you are adding text directly into your document from the keyboard and you can maneuver the text area using the arrow keys.

For starting a new document, simply type vi after the command prompt to the start the vi editor. The cursor will be located in the top left corner and each of the following lines will start with a tilde (~) denoting empty lines. Note that the vi editor in currently in command mode. In order to edit an existing file, type the name of the file along with the extension after typing vi and a space in the Unix command prompt. To enter into the insert mode simply press a on the keyboard. Then to save the edited document, type a colon (:) and the cursor will be located in the bottom left corner after the colon, followed by a w and then press enter. In order to quit the vi editor type a colon followed by a q (:q), this will take you back to the Unix prompt. You can also choose to quit editing without saving by typing :q!, and if you forget the exclamation mark (!) the system will issue a warning.

Note: Do not forget to change the permission settings for the edited file. This is done from the Unix command prompt by typing chmod ugo+wx filename.extension. In ugo, u = user, g = group, o = other, and in wrx, w = write, r = read, x = execute, and the above command can be interpreted as, the user, the group and everyone else can read, write and execute the file. To
remove permissions, just replace the plus sign (+) with a minus sign (-). The .tcl file must be executable.

**Example**

The following script defines a simple topology of four nodes, and two agents, a UDP agent with a CBR (Constant Bit Rate) traffic generator, and a TCP agent. The simulation runs for 3 seconds. The output is two trace files, intro1.tr and intro1.nam. When the simulation completes at the end of 3s, it will attempt to run a nam (nam is a Tcl/Tk based animation tool for viewing network simulation traces and real world packet trace data) visualization of the simulation on your screen.
set ns [new Simulator] ;     # initialize the simulation;
set f [open out.tr w ] ;     # Predefine tracing
$ns trace-all $f
set nf [open out.nam w ]
$ns namtrace-all $nf
set n0 [$ns node] ;          # initialize the nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link s0 n2 5Mb 2ms DropTail ;   # DropTail == FIFO
$ns duplex-link s1 n2 5Mb 2ms DropTail
$ns duplex-link s2 n3 1.5Mb 10ms DropTail

set udp0 [new Agent/UDP] ;     # Initialize an UDP Agent, $udp0;
$ns attach-agent $n0 $udp0 ;     # Attach UDP agent udp0 to Node $n0;
set cbr0 [new Application/Traffic/CBR] ;     # Initialize a CBR traffic generator agent;
$cbr0 attach-agent $udp0 ;     # Attach UDP Agent, $udp0, to CBR Traf. Gen., $cbr0;
$udp0 set class_ 0 ;     # actually, the default, but....;

set null0 [new Agent/Null] ;     # $null0 is a sink;
$ns attach-agent $n3 $null0 ;     # Attach Node $n3 to the Sink $null0;
$ns connect $udp0 $null0 ;     # Make $n3 a sink for UDP Agent, udp0;
$ns at 1.0 "$cbr0 start" ;     # At time 1.0 sec start $cbr0;
pus [ $cbr0 set packetSize_ ] ;     # Displays default packetSize on-screen
puts [ $cbr0 set interval_ ] ;     # Displays default interval on-screen

; # A FTP over TCP/Tahoe from $n1 to $n3, flowid 2
set tcp [new Agent/TCP]
$tcp set class_ 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink

set ftp [new Application/FTP] ;     # TCP does not generate its own traffic;
$ftp attach-agent $tcp
$ns at 1.2 "$ftp start"
$ns connect $tcp $sink
$ns at 1.35 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

# The simulation runs for 3s.
# The simulation comes to an end when the scheduler invokes the finish() procedure below.
# This procedure closes all trace files, and invokes nam visualization on one of the trace files.
$ns at 3.0 "finish"
proc finish {} {
    global ns f nf
    $ns flush-trace
dose $f
dose $nf
    puts "running nam..."
    exec nam out.nam &
exit 0
}

# Finally, start the simulation.
$ns run
Upon running the above file, a trace file, intro1.tr, is generated and the following section discusses the format of the trace file. The default parameters can be found in ns/tcl/lib/ns-default.tcl (you might have to download ns-allinone-2.1b7a.

**Trace File**

Upon running a simulation, different variables can be traced which can then be used for extracting useful information. On activating 'trace-all', all traceable variables are recorded in the output trace file. A sample row from the resulting trace file is as follows:

```
   r  1.006086  0  2  dbr  210  - - - - - -  0  0.0  3.1  1  1
```

The results invoked by a **trace-all** command consists of 11 Fields and 7 Flags, and the description of which are as follows:

- **Field 1:** Type of action performed:
  - a) `'+' for an enqueue operation,
  - b) `'-' for a dequeue operation,
  - c) `'r' for receive packets,
  - d) `'d' for drop packets.

- **Field 2:** Simulated time (seconds).
- **Field 3:** Starting Node.
- **Field 4:** Ending Node.
- **Field 5:** Type of packet (such as TCP, UDP, CRB). For further listing, please refer ns2 manual.
- **Field 6:** Packet Size (bytes).
- **Field 7:** The IP flow identifier as defined in IP version 6.
- **Field 8:** Source node address.
- **Field 9:** Destination node address.
- **Field 10:** Sequence number. In NS version 1, all packets contained a sequence number, whereas in NS version 2 only those agents interested in providing sequencing will generate sequence number. Thus, this field may not be useful in NS v2 for packets generated by agents that have not filled in a sequence number. It is used here to remain backward compatible.
- **Field 11:** Each new packet created in the simulation is assigned a new unique identifier.

The trace file contains useful data from the simulation, and using these results one can find the number of the packets dropped, the throughput, etc. So, in order to parse out information from the trace file we will use awk, which is another scripting language.

**Awk**

The awk program is specially designed for working with structured files and text patterns. It has built-in features for breaking input lines into fields and comparing these fields to patterns that you specified.

Let’s say we are interested in knowing:

1. The Total # of bytes transferred via TCP.
2. The Total # of bytes transferred via CBR.
3. The Total # of bytes transferred via either TCP/CBR.
4. Ratio of TCP and TCP/CBR traffic.
5. Ratio of CBR and TCP/CBR traffic.

In order to generate the above results, here are the steps:

1. Create a file named intro1c and enter the following in your file
   
   ```
   if ($1 == "r" && $3 == 2 && $4 == 3 && $5 == "tcp") {a += $6} else if ($1 == "r" &&
   $3 == 2 && $4 == 3 && $5 == "cbr") {b += $6} END {printf (" tcp:\t%d
 cbr:\t%d

 total:\t%d\n tcp ratio:\t%.4f \n cbr ratio:\t%.4f\n", a, b, a+b, a/(a+b), b/(a+b))}
   ```

2. The type the following by the Unix command prompt:
   ```
   awk -f intro1c intro.tr
   ```

3. The above will generate the following results:
   ```
   tcp: 23000
   cbr: 111300
   total: 134300
   tcp ratio: 0.1713
   cbr ratio: 0.8287
   ```

In step 1, $1 denotes the value of the first field. The if/elseif statement
is repeated for every row of data in the trace file, intro1c.tr, thus the END
pattern applies after the last line read. In step 2, the -f tells awk that the next
filename is a program file rather than an input file and the following command
runs a program saved in a file called intro1c on the contents of intro1.tr. Please
refer Appendix B for some addition material on awk.

References:
1. NS2 Archive: http://www.isi.edu/nsnam/ns
2. NS2 Tutorial: http://www.isi.edu/nsnam/ns/tutorial
APPENDIX A
(vi)
APPENDIX B