Experimental Networking

Lab 2, Network Simulator ns2

Shiv Kalyanaraman
Yong Xia (TA)
Satish Raghunath

shivkuma@ecse.rpi.edu
http://www.ecse.rpi.edu/Homepages/shivkuma

Sept 4, 2003
Ns2 Tutorial

- Languages
- Ns2
- Nam
- Assignment #2
Nam View
Languages

- **System language: C, C++, Java**
  - Build data structures and algorithms from scratch
  - Strongly typed to manage complexity
  - Compiled, high efficiency, 10~20x faster

- **Scripting language: Perl, Tcl(OTcl), Unix shell**
  - Rapid high level programming: to “glue” applications
  - Typeless to simplify connections between components
  - Interpreted, less efficient
  - Sacrifice execution speed for development speed
    (5~10x faster than system language for gluing dev)
Ns2

• NS is a “Network Simulator”
  - Can setup network topologies
  - Generate packet traffic similar to Internet and measure various parameters

• NS is needed because:
  - Need to verify utility / feasibility of new algorithms / architectures
  - Actual topologies Expensive / Error prone / Time consuming to setup
Ns2 status

• Source code
  – C++ for packet processing, Otcl for control
  – 100K lines of C++; 70K lines of OTcl; 50K+ lines of test suite, examples, docs
  – http://www.isi.edu/nsnam/ns/
  – Current version 2.26, (v2.1-b5 installed)

• Platforms
  – Most UNIX systems (FreeBSD, Linux, Solaris)
  – Window 9x/NT/2000
Ns2

• Most of the NS2 source code is in C++
  – http://www.isi.edu/nsnam/ns
  – http://www.isi.edu/nsnam/ns/ns-documentation.html
• Tcl is a simple scripting language used in conjunction with Otcl to setup simulation topologies and scenarios.
  – http://dev.scriptics.com/man/tcl8.2.3/
• OTcl adds Object orientation to Tcl
  – http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/tutorial.html
• NAM – Network Animator is used to visualize simulations
  – http://www.isi.edu/nsnam/nam
Tcl

- `expr 20 + 10`
- `set x 32`
- `set cmd expr; set x 11; $cmd $x*$x`
- `set a 44; set b [expr $a*4]`
- `set x 24; set y 18; set z "$x + $y is [expr $x + $y]"
- `set x 24; set y 18; set z {$x + $y is [expr $x + $y]}`
- `proc power {base p} {
    set result 1
    while {$p > 0} {
        set result [expr $result * $base]
        set p [expr $p - 1]
    }
    return $result
}
Anatomy of a simple Tcl Script

- Examine “simple.tcl”

  # This is a simple Tcl script to illustrate basic operations
  puts “Executing simple tcl script”
  # Open a file for writing
  set f1 [open “try” “w”]
  # Write something into the file and close it
  puts $f1 “Writing a sentence into file”
  close $f1
  # Read the sentence
  set f1 [open “try” “r”]
  set l1 [gets $f1]
  puts “Read line: $l1”
You can run the Tcl script using the program “tclsh” as:

```bash
~> tclsh simple.tcl
Executing simple tcl script
Read line: Writing a sentence into file
~>
```

Let us observe the syntax:
- Lines beginning with “#” are treated as comments
- The symbol “$” is used to obtain the contents of a variable
- The “set” method is used to assign values to variables. Note that the “$” does not appear when something is being assigned to the variable
- The effect of parentheses in math is obtained by using [ ] – e.g., [open ...] indicates that the code in the brackets is evaluated first and then assigned to f1
- “puts”, “gets”, “open” are all Tcl commands. “puts $f1 ...” indicates that we are passing contents of f1 as a parameter to puts

Simple.tcl thus opens a file called “try”, writes a sentence and reads from it
Exercise 1- Loops and Lists

- This exercise will introduce you to loops and lists in Tcl.
- A file contains information about path in a network. The path is specified as a list of numbers: 1 15 7 25 3 25 2 10 5 ... to indicate that node 1 is connected to node 7 with a link of 15Mbps, node 7 to node 3 with 25Mbps and so on. Write a tcl script to read this file and interpret its contents. Your output should look like:

Link 1: Node 1 to Node 7; Bandwidth: 15M
Link 2: Node 7 to Node 3; Bandwidth: 25M

- You might want to use the following functions:

  # Create an empty list in l1
  set l1 [list]

  # Concatenate a list/string with another and assign to l2
  set l2 [concat $l1 $s1]

  # Access an element i from the list l2
  set el1 [lindex $l2 $i]

  # Execute a statement n times
  for {set i 0} { $i < n } {incr i} {
    ...
  }
Tcl

- Stop here, let students do Tcl program
Otcl Examples

- A “class” is like a struct with facilities for private and public variables, member functions and inheritance
- Let's examine the topology class:

```otcl
Class Topology

Topology instproc init { } {
    $self instvar nodes ns
    set ns [Simulator instance]
    set nodes(1) [$ns node]
    set nodes(2) [$ns node]
    $ns duplex-link $nodes(1) $nodes(2) 10M 10ms DropTail
}

Topology instproc get-node { node-id } {
    return $nodes($node-id)
}
```
Otcl Examples (contd.)

- To understand all aspects in the above example you have to know the basics of Object oriented programming. I will assume that you know at least the meaning of these terms: member functions, static functions/variables, instances, constructors.
- The first line is the declaration of “Topology” as a class.
- The function “init” is the equivalent of constructor in C++ - it is the first function called when an instance of this class is created by “new” operator.
- $self is equivalent to the “this” pointer in C++. It refers to the present instance within which the function is executing – that is it refers to “itself”.
- “instvar” is used to declare a member variable and similarly “instproc” is used to declare a member function. The syntax of a procedure is similar to that in Tcl except that the class name has to come first and the “proc” keyword is replaced by instproc. The empty braces (“{}”) indicate that the procedure takes no parameters.
Otcl Examples (contd.)

- Note that the variable ns is being assigned "[Simulator instance]". "Simulator" is the name of a class. "instance" is a static function in the class which returns the instance of the Simulator class (already in memory).
- The general syntax to access member functions is
  
  $\text{obj member-func parameters}$

  This can be observed where the duplex-link function is called to create a link between nodes(1) and nodes(2).
- nodes() is an array. As noted in the example no special declaration is needed to use arrays.
- To use this class, we may write this code:
  
  ```otcl
  set ns [new Simulator]
  set t1 [new Topology]
  set n1 [$t1 get-node 1]
  ```
OTcl

- Stop here, let students do OTcl program
How ns2 works

OTcl Script Simulation Program

OTcl : Tcl interpreter with OO extension

NS Simulator Library
- Event Scheduler Objects
- Network Component Objects
- Network Setup Helping Modules (Plumbing Modules)

Simulation Results
Analysis

NAM Network Animator
An example: skeleton

- A ns-2 simulation script generally includes
  - Create the event scheduler
  - Turn on tracing, if needed
  - Create network topology
  - Setup routing
  - Create transport agent
  - Create traffic source/sink
  - Transmit application-level data
An example: how to start

- Create a event scheduler
  - set ns [new Simulator]

- Open a file for trace data
  - set nf [open out.nam w]
  - $ns namtrace-all $nf
An example: how to start

- A procedure to close file and start NAM
  - proc finish {} {
    global ns nf
    $ns flush-trace
    close $nf
    exec nam out.nam &
    exit 0
  }

- Schedule the procedure
  - $ns at 5.0 "finish"

- Start simulation
  - $ns run
An example: topology

- **Node**
  - set n0 [$ns node]
  - set n1 [$ns node]
  - set n2 [$ns node]

- **Link**
  - $ns duplex-link $n0 $n1 1Mb 5ms DropTail
  - $ns duplex-link $n1 $n2 400Kb 10ms DropTail
An example: agent / application

- Create a UDP agent and attach it to node n0
  - set udp [new Agent/UDP]
  - $ns attach-agent $n0 $udp

- Create a CBR traffic source and attach it to udp0
  - set cbr [new Application/Traffic/CBR]
  - $cbr attach-agent $udp

- Create a null agent to be traffic sink
  - set null [new Agent/Null]
  - $ns attach-agent $n2 $null
An example: agent / application

- Connect them
  - $ns connect $udp $null

- Schedule the event
  - $ns at 0.5 "$cbr start"
  - $ns at 4.5 "$cbr stop"
An example: agent / application

- Stop here, let students do UDP transmission simulation
An example: agent / application

- Create a TCP agent and attach it to node n0
  - set tcp [new Agent/TCP]
  - $ns attach-agent $n0 $tcp

- Create a FTP traffic source and attach it to udp0
  - set ftp [new Application/FTP]
  - $ftp attach-agent $tcp

- Create a TCPSink agent to be traffic sink
  - set sink [new Agent/TCPSink]
  - $ns attach-agent $n2 $sink
An example: agent / application

- Schedule the event
  - $ns at 0.5 "$ftp start"
  - $ns at 4.5 "$ftp stop"
Traces

- **Traces in NS format**
  - `$ns trace-all [open tr.out w]`

  ```
  <event> <time> <from> <to> <pkt> <size> -- <fid> <src> <dst> <seq> <attr>
  + 1 0 2 cbr 210 -------- 0 0.0 3.1 0 0
  - 1 0 2 cbr 210 -------- 0 0.0 3.1 0 0
  r 1.00234 0 2 cbr 210 -------- 0 0.0 3.1 0 0
  d 1.04218 1 2 cbr 210 -------- 0 0.0 3.1 0 0
  ```

- **Traces in NAM format**
  - `$ns namtrace-all [open tr.nam w]`

- **Turn on tracing on specific links**
  - `$ns trace-queue $n0 $n1`
  - `$ns namtrace-queue $n0 $n1`
An example: agent / application

- Stop here, let students do TCP transmission simulation
More settings: event and queuing

- **Schedule events**
  - `$ns at <time> <event>`
  - `<event>`: any legitimate ns/tcl commands

- **Links and queuing**
  - `$ns duplex-link $n0 $n1 <bandwidth> <delay> <queue_type>`
  - `<queue_type>`: DropTail, RED, CBQ, FQ, SFQ, DRR
More settings: Routing

- **Unicast**
  - $\textit{ns rtproto } <\textit{type}>$
  - $<\textit{type}>$: Static, Session, DV, cost, multi-path

- **Multicast**
  - $\textit{ns multicast}$ (right after \[new Simulator\])
    - or set $\textit{ns } [\textit{new Simulator } – \textit{multicast on}]$
  - $\textit{ns mrtproto } <\textit{type}>$
  - $<\textit{type}>$: CtrMcast, DM, ST, BST (centralized, dense mode, shared tree)
More settings: Traffic on Top of UDP

- **UDP**
  - set udp [new Agent/UDP]
  - set null [new Agent/Null]
  - $ns attach-agent $n0 $udp
  - $ns attach-agent $n1 $null
  - $ns connect $udp $null

- **CBR**
  - set src [new Application/Traffic/CBR]

- **Exponential or Pareto**
  - set src [new Application/Traffic/Exponential]
  - set src [new Application/Traffic/Pareto]
More settings: Traffic on Top of TCP

- **TCP**
  - set tcp [new Agent/TCP]
  - set tcpsink [new Agent/TCPSink]
  - $ns attach-agent $n0 $tcp
  - $ns attach-agent $n1 $tcpsink
  - $ns connect $tcp $tcpsink

- **FTP**
  - set ftp [new Application/FTP]
  - $ftp attach-agent $tcp

- **Telnet**
  - set telnet [new Application/Telnet]
  - $telnet attach-agent $tcp
The slides till now have provided the basics of what is needed to run simulations in NS. To explore further, you can exploit the following sources:

- Example code in `tcl/test` directory of your NS distribution
- NS Mailing lists: [http://www.isi.edu/nsnam/ns/ns-lists.html](http://www.isi.edu/nsnam/ns/ns-lists.html)
Assignment #2

- See assignment page