# Working with NS-2

Ritesh Kumar, UNC-Chapel Hill

## Simulation of Artificial Worlds

- What if…
  - I had large meadows and forests together…
    with wild animals to hunt / eat / wear / show-off…
    with super-powers…
    and Deadly Enemies !
- Good entertainment ☺
- Have to pay to play ☹


## NS-2 is not very different…

- What if…
  - I had a whole bunch of computers here…
    with another whole bunch there…
    with a whole mess of links from here to there…
    and my protagonist Protocol from Stardom !
- Advances Networking Research ☺
- Comes for FREE! ☺

NS-2

## What is NS-2 like?

- No… its not a game, but feel free to be happy…
- NS-2 is a scripting language
  - Derivative of Tcl/Tk
- NS-2 implements a lot of networking stuff
  - Hidden beneath the Tcl/Tk language layer
- NS-2 implements a simulator engine
  - Simulation time ≠ Real (wall clock) time
- NS-2 usage:
  - Write a simulation program script
  - Run it… retrieve and post-process output files
  - Analyze results for hidden gems to trade for degree…

## Lesson 1: Setting up the Environment

Difficulty level: "Easy"

## UNIX PATH setup

- NS-2 (and associated post processing tools) run on UNIX
- Setup your PATH in your UNIX (Linux) environment:

export PATH=/afs/unc/proj/rtt/ns2/bin:$PATH

  - Put this in your .profile/.bashrc
  - Don't forget the ending $PATH !

  - logout and re-login
  - 'which ns' should say /afs/unc/proj/rtt/ns2/bin/ns

# Lesson 2: Understanding NS-2 scripting

Difficulty Level: "Difficult"

## NS-2 scripting

- NS-2 it derived from the Tcl language
  - First few chapters of the following should get you started
  - http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html
- Variables
  - set using the command set <varname> <value>
  - accessed using $varname
- Numeric Computation
  - use the *expr* command and [] to substitute result
    - set oneplusone [expr 1 + 1]

## Conditions and Loops

- if {< condition >} {<then action>} else {<else action>}
- while {<condition>} {<loop action>}
- for {<init>} {<condition>} {<next action>}
  {<loop action>}
  - eg.

```
for {set i 0} {$i < 10} {incr i} {
    puts "i is: $i"
}
```

## Creating simulation "Objects"

- NS-2 scripting is also object-oriented
  - Consult NS-2 manual for all kinds of objects and properties
  - http://isi.edu/nsnam/ns/doc/index.html
- Creating Simulator Object
  - set *ns* [new Simulator]
  - *ns* manages nodes and links
  - *ns* manages simulation time
- Creating nodes
  - set node1 [$ns node]

## Creating simulation "Objects"

- Creating links between nodes
  - $ns duplex-link $node1 $node2 10Mbps 1ms Droptail
  - A bidirectional link is created between $node1 and $node2
  - The link's capacity is 10Mbps
    - Yes, ns-2 requires units for bandwidth and time !
  - The link propagation latency is 1ms
    - Note the units again !
  - We select a simple FIFO Droptail queue-type for the link

## Creating simulation "Objects"

- Creating Agents
  - Agents are sending/receiving units for protocols like TCP/UDP
  - Agents need to be attached to nodes
  - set udp_sender [new Agent/UDP]
    $udp_sender set packetSize_ 1500
    $ns attach-agent $node1 $udp_sender
  - set udp_sink [new Agent/LossMonitor]
    $ns attach-agent $node2 $udp_sink
  - Agents need to be connected
    $ns connect $udp_sender $udp_sink

## Creating simulation "Objects"


- Creating TCP Agents
  - set tcp_sender [new Agent/TCP]
    $tcp set fid_ 2
    $ns attach-agent $node1 $tcp_sender
  - set tcp_sink [new Agent/TCPSink]
    $ns attach-agent $node2 $tcp_sink
- TCP sender's flow id (fid_) shows up in the trace

## Creating simulation "Objects"

- Creating Applications
  - set cbr [new Application/Traffic/CBR]
    $cbr set packetSize_ 1500
    $cbr set rate_ 2Mbps
  - Applications need to be attached to agents
    $cbr attach-agent $udp_sender
  - Sinks in general need not be connected to applications
- You can start the CBR application using "$cbr start"

## Creating simulation "Objects"


- Creating TCP Applications
  - set ftp [new Application/FTP]
    $ftp attach-agent $tcp_sender
- You can start the FTP application using "$ftp start"

## Scheduling simulation events

- Simulation time ≠ Real (wall clock) time
- NS-2 allows you to schedule things at different times
  - $ns at <time in seconds> <expression to evaluate>
  - $ns at 10.0 "$cbr start"
- Starting simulation is easy
  - $ns run
- Let's run our script…

## Monitoring simulation progress

- Let's add a "heart beat" function
  - Well print something every 1 second in simulation time.
  - $ns now gives us the current simulation time.
  - proc heart_beat {} {
    global ns
    set now [$ns now]
    puts stderr "Simulation Time: $now"
    $ns at [expr $now+1] "heart_beat"
    }
  - We schedule the first heart_beat call at time 0.0
- Let's try our new script…

## Putting an end to all this…

- Add a finish function and schedule it
  - proc finish {} {
    exit 0
    }
  - schedule it at a given point in time
    $ns at 30.0 "finish"
- Let's try that again…

## Collecting a trace

- A simulation run is useless if we don't have data to study !
- Adding trace collection
  - Create a handle to a trace file
    - These can become very large hence we use compression.
  - set tracefile [open "| gzip > exp.tr.gz" w]
    $ns trace-queue $node1 $node2 $tracefile
    $ns trace-queue $node2 $node1 $tracefile
  - flush the trace and close the tracefile before exiting in finish.
- Yay! we have data now…

## The trace file format

- <event type> <time> <source> <destination> <packet type> <packet size> --------
  <flow id> <source ip addr> <destination ip address>
  <sequence no.> <unique packet id>

+ 10.006 0 1 cbr 1500 ------- 0 0.0 1.0 1 1

- 10.006 0 1 cbr 1500 ------- 0 0.0 1.0 1 1

r 10.0082 0 1 cbr 1500 ------- 0 0.0 1.0 1 1

Hmm... that's a lot of data.

## Lesson 3: Post processing of data

Difficulty Level: "Very Difficult"

## Example: no. of packets vs. time

- How many packets were seen by a given point in time?
  - Aim is to generate a plot of packets w.r.t. time
- Steps:
  1. Filter out de-queue entries from the trace
  2. Separate out the time column
  3. Add a simple counter column
  4. Plot it
- We use 'awk' for post processing and 'gnuplot' for plots

## Using awk

- Documentation: man awk
  - Tutorials can be found on the web
- Awk basically calls a function on every line of input data
  - seq 1 20 | awk '{print sqrt($1)}'
    - prints the square roots of numbers 1 to 20
  - The fields in the input lines are available as $1, $2, $3, … etc
    - The whole line is available at $0 (good for debugging)
  - Awk can filter the input data on a regular expression
    - awk '/<regexp>/{ <awk code> }'
  - Awk code resembles C except that it doesn't have types.

## Using gnuplot

- Featureful plotting utility
- Supports
  - line styles, point styles, colors
  - log axes
  - auto ranges
  - postscript / eps rendering
- Let's plot our data…

## Lesson 4: Finding Gems after analysis

Difficulty Level: "Ultra Difficult"

## Trading gems for a degree (MS/Ph.D)

- There is no secret sauce…
- Each man on his own…
- Lots of coffee…

- May the force be with you!