NS2 Tutorial
What is a simulation

Simulation
- Imitation of the operation of a facility or process
- Facility being simulated is called a system
- Assumptions/approximations, both logical and mathematical, are made about how the system works
- These assumptions form a model of the system

Simulation allows you
- To quickly & inexpensively acquire knowledge concerning a problem
- That is usually gained through experience (that is often time-consuming)
When to use simulations?

Simulations can be used
- To study complex systems i.e., systems where analytic solutions are infeasible
- To compare design alternatives for a system that doesn't exist
- To study the effect of alterations to an existing system
- To reinforce/verify analytic solutions

Simulations should not be used
- If model assumptions are simple such that mathematical methods can be used to obtain exact answers (analytical solutions)
Simulation Flow

Define problem → Simulation model → Execute simulation → Extend simulator → Post-process results
Types of Simulation

- Variety of types, but main are
  - Monte Carlo Simulations
  - Trace Driven Simulations
  - Discrete-event Simulations
    - A simulation using discrete-event (also called discrete state)
    - model of the system
    - Widely used for studying computer networks
Components of Discrete-Event Simulation

- Typical components:
  1. Event scheduler
  2. Simulation clock
  3. System state variables
  4. Event routines
  5. Input routines
  6. Report generator
What is ns-2?

- ns-2 stands for Network Simulator version 2.
- ns-2 is a discrete event simulator for networking research.
- Work at packet level.
- Provide substantial support to simulate bunch of protocols like TCP, UDP, FTP, HTTP and DSR.
- Simulate wired and wireless network.
- It is primarily Unix based.
- Use TCL as its scripting language.
- ns-2 is a standard experiment environment in research community.
NS-2 Features

- NS-2: network simulator version 2
  - Discrete event simulator
  - Packet level simulation

- Features
  - Open source
  - Scheduling, routing and congestion control
  - Wired networks: P2P links, LAN
  - Wireless networks:
  - terrestrial (ad-hoc, cellular, GPRS, UMTS, WLAN, Bluetooth)
  - satellite
NS-2 Features: cont...

- **Modeling Network Components**
  - Traffic models and applications
    - Web, FTP, telnet, audio, sensor nets
  - Transport protocols
    - TCP (Reno, SACK, etc), UDP, multicast
  - Routing and queueing
    - static routing, DV routing, multicast, ad-hoc routing
    - queueing disciplines: drop-tail, RED, etc
  - Link layer
    - wired, wireless, satellite

- **Providing Infrastructure for**
  - tracing, visualization, error models, etc
  - modify or create your own modules
NS components

- NS, the simulator itself
- NAM, the Network Animator
  - visualize NS (or other) output
- pre-processing:
  - traffic and topology generators
- post-processing:
  - simple trace analysis, often in Awk, Perl, or Tcl
NS Software Structure: C++ and OTCL

- Uses two languages:
  - 1. C++ for the core of NS simulator
     - per packet processing
     - fast to run, detailed, complete control
  - 2. OTCL for control
     - (object Tool command Language)
     - A scripting (interpreted) language
     - simulation setup, configuration
     - fast to write and change
Why two language? (Tcl & C++)

- C++: Detailed protocol simulations require systems programming language
  - byte manipulation, packet processing, algorithm implementation
  - Run time speed is important
  - Turn around time (run simulation, find bug, fix bug, recompile, re-run) is slower

- Tcl: Simulation of slightly varying parameters or configurations
  - Quickly exploring a number of scenarios
  - iteration time (change the model and re-run) is more important
Steps when using NS

- Create OTCL script for your network model
  - nodes, links, traffic sources, sinks, etc.

- Parameterize simulation objects
  - Links: queue sizes, link speeds, ...
  - Transport Protocols: TCP flavor and parameters (more than 30), ...

- Collect statistics
  - dump everything to trace, post process it
  - gather stats during simulation within OTCL script
  - modify Ns source code

- Run NS multiple times
  - confidence intervals
Designing and running simulations in Tcl using the simulator objects in the OTcl library. The event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using tclcl. The whole thing together makes NS, which is a OO extended Tcl interpreter with network simulator libraries.
Tcl: Overview

- Tcl: Tool command language
- Tcl is extensible and embeddable
  - NS-2 is also a Tcl interpreter (tclsh or ns)
- Tcl is a scripting language
  - Ideal for network configuration
- A Tcl script consists of commands
- To write Tcl scripts, you need to learn
  - Tcl command
  - Tcl syntax (how commands are parsed)
Tcl: Command

- A command consists of words
  - `cmdName arg1 arg2 ...
  - `cmdName`: core command or procedure
    - set, puts, expr, open, if, for, ...
  - All words are considered as strings
  - White space (space/tab) separates arguments
  - Newline or semicolon (;) terminates a command

- Command evaluation: parsing and execution
  - The interpreter does "substitution" and "grouping"
  - (parsing) before running a command
  - Every command returns a result string after execution
TCL Overview:

- Similar to other programming languages with some peculiarities
  - variables not declared and are all of generic type
    - `set a 3` instead of `a=3`
  - mathematical operations cumbersome
    - `[expr 3+4]`
      - returns the value 7
  - use "$" to read from a variable
    - `set a $b` instead of `set a b` or `a==b`
      - equates `a` with `b`'s value
  - write output
    - `put "hello world"`
      - outputs `hello world`
    - `put $a`
      - outputs `a`'s value
NS2 installation

Download source code ns-allinone-2.29.tar.gz
[root@localhost Desktop]mkdir test
[root@localhost test]cp ns-allinone-2.29.tar.gz /root/Desktop/test
[root@localhost test]tar -xzvf ns-allinone-2.29.tar.gz
[root@localhost test]cd ns-allinone-2.29
[root@localhost test]./install
[root@localhost test] vi /root/.bash_profile
PATH=$PATH:/home/bin:/root/Desktop/test/ns-allinone/bin:/root/Desktop/test/ns-allinone-2.29/tcl-8.4.14/uni
x:/root/Desktop/test/ns-allinone/tk-8.4.14/uni
Export PATH
Press Esc and Shitt ZZ
Also Export LD_LIBRARY_PATH
Export TCL_LIBRARY according to your system paths
Writing First Procedure (Example 1)

```python
# Writing a procedure called "test"
proc test () {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
        if {$k < 5} {
            puts "$k < 5, pow = [expr pow($d, $k)]"  
        } else {
            puts "$k >= 5, mod = [expr $d % $k]"
        }
    }
}
```

- **proc** is used to define a procedure
- **$** is used with the variable name
- **puts** prints out the following string within double quotation marks
Example 2

Class mom
mom instproc greet {}
    $self instvar age_
puts "$age_ years old mom say:
    How are you doing?"
}
Class kid -superclass mom
kid instproc greet {}
    $self instvar age_
puts "$age_ years old kid say:
    What's up, dude?"
}
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15
$a greet
$b greet

- The keyword Class is to create an object class and instproc is to define a member function to an object class
- Class inheritance is specified using the keyword -superclass
- $self acts same as the "this" pointer in C++,
- to create an object instance, the keyword new is used
Components

- Four major types
  - Application
    - Communication instigator
  - Agent
    - Packet generator/consumer
  - Node
    - Addressable entity
  - Link
    - Set of queues
OTCL hierarchy
Example 03

Network Diagram
Example 03 cont...

```plaintext
# Create a simulator object
set ns [new Simulator]

# (set ns [new Simulator]: generates an NS simulator object instance, and assigns it to variable ns )

# Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

# Open nam trace file (This member function tells the simulator to record simulation traces in NAM input forma)
set nf [open out.nam w]
$ns namtrace-all $nf

# Define a finish procedure
Proc finish ()
{
    global ns nf
$ns flust_trace
```
Example 03 cont...

#close the namtrace 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
Example 03 cont...

```plaintext
# 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 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 # marking tcp flow
$ns attach-agent $n0 $tcp
Set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_1 # identifying it in flow 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
Example 03 cont...

```plaintext
# 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"
```
Example 03 cont...

# 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
Executing Script

[root@localhost Desktop]cd test
[root@localhost test]ls
ns-simple.tcl
[root@localhost test] ns ns-simple.tcl
#here simulation will start and nam will appear on screen. Two files will also appear in present working directory with the name of out.nam and out1.tr. out.nam is network amulation file and out1.tr is a trace file containing data about simulation
[root@localhost test]ls
ns-simple.tcl out.nam out1.tr
[root@localhost test]] kedit out.nam out1.tr
Network Amulator (NAM)
### .tr Trace File

```
+ 0.1 0 2 cbr 1000 -------yyy 0.0 3.0 0 0
- 0.1 0 2 cbr 1000 ------- 1 0.0 3.0 0 0
+ 0.1 1 2 tcp 40 ------- 2 1.0 3.1 0 1
- 0.1 1 2 tcp 40 ------- 2 1.0 3.1 0 1
r 0.10532 1 2 tcp 40 ------- 2 1.0 3.1 0 1
+ 0.10532 2 3 tcp 40 ------- 2 1.0 3.1 0 1
- 0.10532 2 3 tcp 40 ------- 2 1.0 3.1 0 1
r 0.113 0 2 cbr 1000 ------- 1 0.0 3.0 0 0
+ 0.113 2 3 cbr 1000 ------- 1 0.0 3.0 0 0
- 0.113 2 3 cbr 1000 ------- 1 0.0 3.0 0 0
r 0.115853 2 3 tcp 40 ------- 2 1.0 3.1 0 1
+ 0.115853 3 2 ack 40 ------- 2 3.1 1.0 0 2
- 0.115853 3 2 ack 40 ------- 2 3.1 1.0 0 2
+ 0.117857 0 2 cbr 1000 ------- 1 0.0 3.0 1 3
- 0.117857 0 2 cbr 1000 ------- 1 0.0 3.0 1 3
r 0.126387 3 2 ack 40 ------- 2 3.1 1.0 0 2
```
### Trace File Cont...

<table>
<thead>
<tr>
<th>event</th>
<th>time</th>
<th>from node</th>
<th>to node</th>
<th>pkt type</th>
<th>pkt size</th>
<th>flags</th>
<th>fid</th>
<th>src addr</th>
<th>dst addr</th>
<th>seq num</th>
<th>pkt id</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>1.3556</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>1.3556</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>1.3556</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>1.3556</td>
<td>0</td>
<td>2</td>
<td>tcp</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source address: node.port (3.0)
Destination address: node.port (0.0)
Abbreviation NS2

- Some abbreviations used:
  - Event:
    - S=send
    - R=receive
    - D=drop
    - F=forward
  - Trace level:
    - AGT=agent
    - RTR=router
    - MAC=MAC
    - IFQ=drop due to full buffer
    - ARP=drop by ARP
Acquiring useful data from .tr file

- `cat out.tr | grep " 2 3 cbr " | grep ^r | column 1 10 | awk '{dif = $2 - old2; if(dif==0) dif = 1; if(dif > 0) printf("%d\t%f\n", $2, ($1 - old1) / dif); old1 = $1; old2 = $2}') > jitter.txt`

  This shell command selects the "CBR packet receive" event at n3, selects time (column 1) and sequence number (column 10), and calculates the difference from last packet receive time divided by difference in sequence number (for loss packets) for each sequence number.

- Data will be written to jitter.txt
XGraph:

- NS-2 relies on XGraph in order to display simulation results
- XGraph is a UNIX graphing application
NS2 Useful Links

- Network Simulator NS-2
  - NS-2 Homepage (http://www.isi.edu/nsnam/ns/)
  - NS-2 Manual (http://www.isi.edu/nsnam/ns/nsdocumentation.html)

- Getting started with NS-2
  - Marc Greis's tutorial (http://www.isi.edu/nsnam/ns/tutorial/index.html)
  - NS by example (http://nile.wpi.edu/NS/)