CS551
NS Tutorial
Slides Developed by:
John Heidemann
johnh@isi.edu
USC/ISI
ns-2, the network simulator

- a discrete event simulator
  - simple model

- focused on modeling network protocols
  - wired, wireless, satellite
  - TCP, UDP, multicast, unicast
  - web, telnet, ftp
  - ad hoc routing, sensor networks
  - infrastructure: stats, tracing, error models, etc.
ns goals

- support networking research and education
  - protocol design, traffic studies, etc.
  - protocol comparison

- provide a collaborative environment
  - freely distributed, open source

- share code, protocols, models, etc.
  - allow easy comparison of similar protocols
  - increase confidence in results

- more people look at models in more situations

- experts develop models

- multiple levels of detail in one simulator
ns history

- Began as REAL in 1989
- ns by Floyd and McCanne at LBL
- ns-2 by McCanne and the VINT project (LBL, PARC, UCB, USC/ISI)
- currently maintained at USC/ISI, with input from Floyd et al.
"ns" components

- ns, the simulator itself
- nam, the Network AniMator
  - visualize ns (or other) output
  - GUI input simple ns scenarios
- pre-processing:
  - traffic and topology generators
- post-processing:
  - simple trace analysis, often in Awk, Perl, or Tcl
ns models

Traffic models and applications:
- web, FTP, telnet, constant-bit rate, Real Audio

Transport protocols:
- unicast: TCP (Reno, Vegas, etc.), UDP
- multicast: SRM

Routing and queueing:
- wired routing, ad hoc rtg and directed diffusion
- queueing protocols: RED, drop-tail, etc.

Physical media:
- wired (point-to-point, LANs), wireless (multiple propagation models), satellite
ns status

- platforms: basically all Unix and Windows
- size: about 200k loc each C++ and Tcl, 350 page manual
- user-base: >1k institutions, >10k users
- releases about every 6 months, plus daily snapshots
Outlines

- **Concepts**
- Essentials
- Getting Started
- Fundamental tcl, otcl and ns
Discrete Event Simulation

- model world as events
  - simulator has list of events
  - process: take next one, run it, until done
  - each event happens in an instant of virtual (simulated) time, but takes an arbitrary amount of real time

- ns uses simple model: single thread of control => no locking or race conditions to worry about (very easy)
Consider two nodes on an Ethernet:

**Simple Queuing Model:**
- $t=1$: A enqueues pkt on LAN
- $t=1.01$: LAN dequeues pkt and triggers B

**Detailed CSMA/CD Model:**
- $t=1.0$: A sends pkt to NIC, A’s NIC starts carrier sense
- $t=1.005$: A’s NIC concludes cs, starts tx
- $t=1.006$: B’s NIC begins receiving pkt
- $t=1.01$: B’s NIC concludes pkt, B’s NIC passes pkt to app
Discrete Event Scheduler

Four types of scheduler:
- List: simple linked list, order-preserving, O(N)
- Heap: O(logN)
- Calendar: hash-based, fastest, default, O(1)
- Real-time: subclass of list, sync with real-time, O(N)
ns Software Structure: object orientation

Object oriented:
- lots of code reuse (ex. TCP + TCP variants)

Some important objects:
- NsObject: has recv() method
- Connector: has target() and drop()
- BiConnector: uptarget() & downtarget()}
ns Software Structure: C++ and Otcl

- Uses two languages
  - C++ for packet-processing
    - fast to run, detailed, complete control
  - OTcl for control
    - simulation setup, configuration, occasional actions
    - fast to write and change
- pros: trade-off running vs. writing speed, powerful/documentated config language
- cons: two languages to learn and debug in
OTcl and C++: The Duality

OTcl (object variant of Tcl) and C++ share class hierarchy.

TclCL is glue library that makes it easy to share functions, variables, etc.
Outlines

- Concepts
- Essentials
- Getting Started
- Fundamental tcl, otcl and ns
Installation and Documentation

- download ns-allinone (if you have your own machine, do not build this on USC servers)
- includes Tcl, OTcl, TclCL, ns, nam, etc.
- run ns and nam on ISD machines:
  - ~csci551/ns
  - ~csci551/nam

- mailing list: ns-users@isi.edu

- documentation (see url above)
  - Marc Gries tutorial
  - ns manual
Hello World

simple.tcl:

```tcl
set ns [new Simulator]
$ns at 1 \"puts \"Hello World!\"\"
$ns at 1.5 \"exit\"
$ns run
```

nunki 74% ~csci551/ns simple.tcl
Hello World!
nunki 75%

Think C++:

```cpp
Simulator *ns=new Simulator;

ns->at(1, \"puts \"Hello World!\"\")
ns->at(1.5, \"exit\")
ns->run();
```
Hello World, Deconstructed

```euclid
set ns [new Simulator]
create a simulator, put in var ns
$ns at 1 "puts "Hello World!"
schedule an event at time t=1 to print HW
$ns at 1.5 "exit"
and exit at a later time
$ns run
run time simulator
```
Outlines

- Concepts
- Essentials
- Getting Started
- *Fundamental tcl, otcl and ns*
Basic Tcl

variables:

```tcl
set x 10
puts "x is $x"
```

functions and expressions:

```tcl
set y [pow x 2]
set y [expr x*x]
```

control flow:

```tcl
if {$x > 0} { return $x } else { return [expr -$x] } while { $x > 0 } {
    puts $x
    incr x -1
}
```

procedures:

```tcl
proc pow {x n} {
    if {$n == 1} { return $x }
    set part [pow x [expr $n-1]]
    return [expr $x*$part]
}
```

Also lists, associative arrays, etc.

⇒ can use a real programming language to build network topologies, traffic models, etc.
Basic otcl

Class Person
# constructor:
Person instproc init {age} {
    $self instvar age_
    set age_ $age
}
# method:
Person instproc greet {} {
    $self instvar age_
    puts "$age_ years old: How are you doing?"
}
# subclass:
Class Kid - superclass Person
Kid instproc greet {} {
    $self instvar age_
    puts "$age_ years old kid: What’s up, dude?"
}

set a [new Person 45]
set b [new Kid 15]
$a greet
$b greet

can easily make variations of existing things (TCP, TCP/Reno)
Creating the event scheduler

[Turn on tracing]

Creating network

Setting up routes

Inserting errors

Creating transport connection

Create traffic
Creating Event Scheduler

Create scheduler
- `set ns [new Simulator]`

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

Start scheduler
- `$ns run`
Creating Network

Nodes

- set n0 [ns node]
- set n1 [ns node]

Links & Queuing

- ns duplex-link $n0 $n1 <bandwidth> <delay> <queue_type>
- <queue_type>: DropTail, RED, CBQ, FQ, SFQ, DRR
Computing routes

- **Unicast**
  - `$ns rtproto <type>`
  - `<type>`: Static, Session, DV, cost, multi-path

- **Multicast**
  - `$ns multicast`
  - right after [new Simulator]
  - `$ns mrtproto <type>`
  - `<type>`: CtrMcast, DM, ST, BST
Traffic

- simple two layers: transport and app
- transports:
  - TCP, UDP, etc.
- applications: (agents)
  - ftp, telnet, etc.
Creating Connection: UDP

source and sink

- set usrc [new Agent/UDP]
- set udst [new Agent/NULL]

connect them to nodes, then each other

- $ns attach-agent $n0 $usrc
- $ns attach-agent $n1 $udst
- $ns connect $usrc $udst
Creating Connection: TCP

source and sink

- set tsrc [new Agent/TCP]
- set tdst [new Agent/TCPSink]

connect to nodes and each other

- $ns attach-agent $n0 $tsrc
- $ns attach-agent $n1 $tdst
- $ns connect $tsrc $tdst
Creating Traffic: On Top of TCP

**FTP**
- set ftp [new Application/FTP]
- $ftp attach-agent $tsrc
- $ns at <time> "$ftp start"

**Telnet**
- set telnet [new Application/Telnet]
- $telnet attach-agent $tsrc
Creating Traffic: On Top of UDP

**CBR**

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

**Exponential or Pareto on-off**

- `set src [new Application/Traffic/Exponential]`
- `set src [new Application/Traffic/Pareto]`
Creating Traffic: Trace Driven

Trace driven

- set tfile [new Tracefile]
- $tfile filename <file>
- set src [new Application/Traffic/Trace]
- $src attach-tracefile $tfile

<file>:
- Binary format
- inter-packet time (msec) and packet size (byte)
Compare to Real World

- more abstract (much simpler):
  - no addresses, just global variables
  - connect them rather than name lookup/bind/listen/accept

- easy to change implementation
  - set tsrc2 [new Agent/TCP/Newreno]
  - set tsrc3 [new Agent/TCP/Vegas]
Inserting Errors

Creating Error Module

- `set loss_module [new ErrorModel]
- `$loss_module set rate_ 0.01`
- `$loss_module unit pkt`
- `$loss_module ranvar [new RandomVariable/Uniform]`
- `$loss_module drop-target [new Agent/Null]`

Inserting Error Module

- `$ns lossmodel $loss_module $n0 $n1`
Tracing

Adobe PDF Document

Trace packets on all links into test.out

$ ns trace-all [open test.out w]

<event> <time> <from> <to> <pkt> <size>--<flowid> <src> <dst> <seqno> <aseqno>

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Packet Type</th>
<th>Flow ID</th>
<th>Source IP</th>
<th>Destination IP</th>
<th>Sequence</th>
<th>Acknowledgement</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>cbr</td>
<td>210</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>3.1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>cbr</td>
<td>210</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>3.1</td>
</tr>
<tr>
<td>r</td>
<td>1.00234</td>
<td>0</td>
<td>2</td>
<td>cbr</td>
<td>210</td>
<td>0</td>
<td>0.0</td>
<td>0</td>
<td>3.1</td>
</tr>
</tbody>
</table>

$ ns namtrace-all [open test.nam w]

<event> can be + for enqueue, - for dequeue, r for receive, d for drop, and e for error
Plumbing: Packet Flow

Diagram showing network nodes n0 and n1 with packet flow through different classifiers and ports, including an entry point and a link n0-n1.
Plumbing: Packet Flow

Diagram showing network nodes n0 and n1 with various classifiers and addresses for packets flows.
Plumbing: Packet Flow

Diagram showing network nodes n0 and n1 with classifier and port details for packets flowing through links n0-n1 and n1-n0.
Plumbing: Packet Flow

Diagram showing the flow of packets between two nodes, n0 and n1, with classifier and port details for different application flows.
Plumbing: Packet Flow

Diagram showing network nodes and data flow with classification and port details.
Plumbing: Packet Flow

- **Link n0-n1**:
  - Entry 0
  - Port Classifier: dst = 1.0
  - Addr Classifier
  - Application/FTP

- **Link n1-n0**:
  - Entry 0
  - Port Classifier: dst = 0.0
  - Addr Classifier
  - Agent/TCP
Plumbing: Packet Flow

- Link n0-n1
- Link n1-n0
- Application/FTP
- Port Classifier
- Addr Classifier
- Agent/TCP
dst_=1.0
dst_=0.0
Plumbing: Packet Flow
Summary: Generic Script Structure

```bash
set ns [new Simulator]
# [Turn on tracing]
# Create topology
# Setup packet loss, link dynamics
# Create routing agents
# Create:
#  - multicast groups
#  - protocol agents
#  - application and/or setup traffic sources
# Post-processing procs
# Start simulation
```
Example - TCP

Simple scenario with TCP and UDP connections
TCP : Step 1

Scheduler & tracing

```tcl
#Create scheduler
set ns [new Simulator]

#Turn on tracing
set f [open out.tr w]
$ns trace-all $f

set nf [open out.nam w]
$ns namtrace-all $nf
```
TCP : Step 2

Create topology

```bash
# create nodes
set n0 [$ns node]
set n1 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
```
#create links
$ns duplex-link $n0 $n1 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n2 $n3 5Mb 2ms DropTail
$ns queue-limit $n1 $n2 25
$ns queue-limit $n2 $n1 25
TCP: Step 4

Create TCP agents

```plaintext
set tcp [new Agent/TCP]
set sink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
```
TCP : Step 5

Attach traffic

```sh
set ftp [new Application/FTP]
$ftp attach-agent $tcp
#start application traffic
$ns at 1.1 "|$ftp start"
```
TCP : Step 6

End of simulation wrapper (as usual)

$ns at 2.0 "finish"
Proc finish {} {
    global ns f
    close $f
    close $nf
    puts "Running nam..."
    exec nam out.nam &
    exit 0
}
$ns run
Viz Tools

- Nam-1 (Network AniMator Version 1)
  - Packet-level animation
  - Well-supported by ns

- Xgraph
  - Convert trace output into xgraph format
Ns-nam Interface

- Color
- Node manipulation
- Link manipulation
- Topology layout
- Protocol state
- Misc
Nam Interface: Color

Color mapping

$ns$ color 40 red
$ns$ color 41 blue
$ns$ color 42 chocolate

Color ↔ flow id association

$tcp0$ set fid_ 40 ;# red packets
$tcp1$ set fid_ 41 ;# blue packets
Nam Interface: Nodes

- **Color**
  
  ```
  $node color red
  ```

- **Shape (can’t be changed after sim starts)**
  
  ```
  $node shape box ;# circle, box, hexagon
  ```

- **Marks (concentric shapes)**
  
  ```
  $ns at 1.0 "$n0 add-mark m0 blue box"
  $ns at 2.0 "$n0 delete-mark m0"
  ```

- **Label (single string)**
  
  ```
  $ns at 1.1 "$n0 label "web cache 0 \"
  ```
Nam Interfaces: Links

- **Color**
  
  ```
  $\text{ns duplex-link-op } n0 n1 \text{ color "green"}
  ```

- **Label**
  
  ```
  $\text{ns duplex-link-op } n0 n1 \text{ label "abcd"}
  ```

- **Dynamics (automatically handled)**
  
  ```
  $\text{ns rtmodel Deterministic \{2.0 0.9 0.1\} } n0 n1
  ```

- **Asymmetric links not allowed**
Nam Interface: Topo Layout

Manual layout: specify everything

\$ns\ \text{duplex-link-op}\ \$n(0)\ \$n(1)\ \text{orient right}
\$ns\ \text{duplex-link-op}\ \$n(1)\ \$n(2)\ \text{orient right-up}
\$ns\ \text{duplex-link-op}\ \$n(2)\ \$n(3)\ \text{orient down}
\$ns\ \text{duplex-link-op}\ \$n(3)\ \$n(4)\ \text{orient 60deg}

If anything missing \textarrow{\rightarrow} \text{automatic layout}
Nam Interface: Protocol State

Monitor values of agent variables

$ns add-agent-trace $srm0 srm_agent0
$ns monitor-agent-trace $srm0
$srm0 tracevar C1_
$srm0 tracevar C2_
# ... ...
$ns delete-agent-trace $tcp1
Nam Interface: Misc

Annotation

- Add textual explanation to your sim

  \$ns at 3.5 \"$ns trace-annotate \"packet drop\"\"

Set animation rate

  \$ns at 0.0 \"$ns set-animation-rate 0.1ms\"
Other Utilities in Ns

- **Nam editor**
  - Available as part of nam-1

- **Tcl debugger**
  - For source and documentation, see
    - [http://www.isi.edu/nsnam/ns/ns-debugging.html](http://www.isi.edu/nsnam/ns/ns-debugging.html)

- **Topology generator**
  - [http://www.isi.edu/nsnam/ns/ns-topogen.html](http://www.isi.edu/nsnam/ns/ns-topogen.html)

- **Scenario generator**
  - [http://www.isi.edu/nsnam/ns/ns-scengeneration.html](http://www.isi.edu/nsnam/ns/ns-scengeneration.html)
Other Ns Features

- Other areas in wired domain
  - LANs
  - Diffserv
  - Multicast
  - Full TCP
  - Applications like web-caching

- Wireless domain
  - Ad hoc routing
  - Mobile IP
  - Satellite networking
  - Directed diffusion (sensor networks)
Other Ns Features

Emulator

- Connect simulator in a real network
- Can receive and send out live packets from/into the real world
Resources

- Ns distribution download
  - http://www.isi.edu/nsnam/ns/ns-build.html

- Installation problems and bug-fix
  - http://www.isi.edu/nsnam/ns/ns-problems.html

- Ns-users mailing list
  - Ns-users@isi.edu
  - See http://www.isi.edu/nsnam/ns/ns-lists.html
  - Archives from above URL
Resources (cont...)

- Marc Greis’ tutorial
  - http://www.isi.edu/nsnam/ns/tutorial

- Ns-users archive

- Ns-manual
  - http://www.isi.edu/nsnam/ns/nsdocumentation.html

- Tcl (Tool Command Language)
  - http://dev.scriptics.com/scripting
  - Practical programming in Tcl and Tk, Brent Welch

- Otcl (MIT Object Tcl)
  - ~otcl/doc/tutorial.html (in distribution)