TELE302 Lecture 5

NS-2

Jeremiah Deng

University of Otago

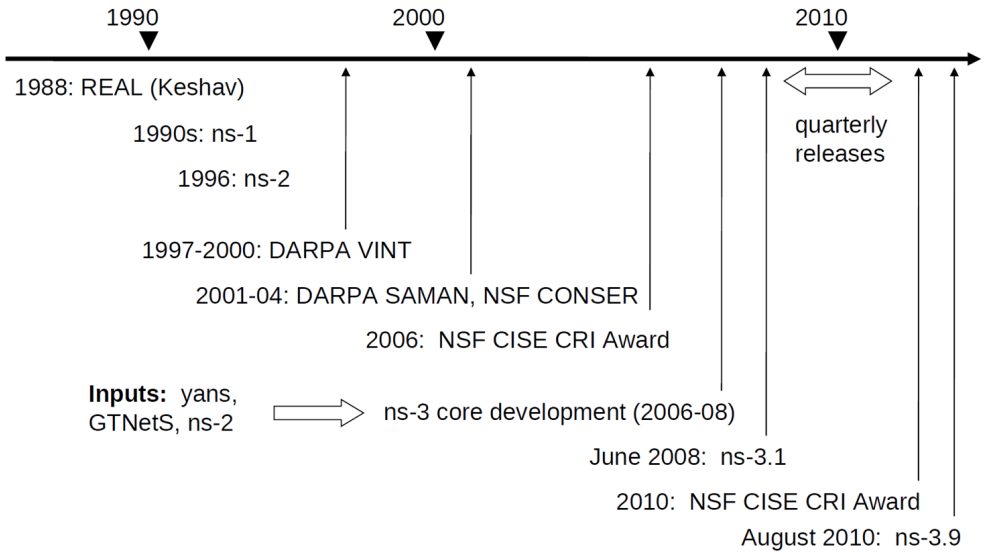
21 July 2015

Lecture Outline

A Bit of History

- 1989: REAL network simulator
- 1995: DARPA VINT project at LBL, Xerox PARC, UCB, and USC/ISI
- Present: DARPA SAMAN project and NSF CONSER project
- Includes substantal contributions from other researchers, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems.

NS Timeline



NS Goals

- Support networking research and education
  - Protocol design, traffic studies, ...
  - Protocol comparison
- Provide a collaborative environment
  - Freely distributed, open source
    - Share code, protocols, models, ...
  - 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

Status

- Periodical release
  - 200K LOC in C++ and Otcl
  - 100 test suites and 100+ examples
  - 431 pages of ns manual
  - Daily snapshot (with auto-validation)
- Stability validation
  - `http://www.isi.edu/nsnam/ns/ns-tests.html`
- Platform support
  - FreeBSD, Linux, Solaris, Windows and Mac

Functionalities

- Wired world
  - Routing: DV, LS, PIM-SM
  - Transportation: TCP and UDP
  - Traffic sources: web, ftp, telnet, cbr, stochastic
  - Queuing disciplines: Drop-tail, RED, FQ, SFQ, DRR
  - QoS: IntServ and Diffserv
  - Emulation
- Wireless
  - Ad hoc routing and mobile IP
  - Directed diffusion, sensor-MAC
- Tracing, visualization, various utilities

Components

- 'ns', the simulator itself
- 'nam', the network animator
  - Visualize *ns* (or other) output
  - Nam editor: GUI interface to generate ns scripts
- Pre-processing:
  - Traffic and topology generators
- Post-processing:
  - Simple trace analysis, often using Awk, Perl, or Tcl

Installation

- Getting the pieces
  - Tcl/TK 8.x
    - `http://resource.tcl.tk/resource/software/tcltk/`
  - Otcl and TclCL:
    - `http://otcl-tclcl.sourceforge.net`
  - ns-2 and nam-1:
    - `http://www.isi.edu/nsnam/dist`
- Other utilities
  - `http://www.isi.edu/nsnam/ns/ns-build.html`
  - Tcl-debug, xgraph, ...

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, easy to implement
  - ☹ no support to parallelism, longer to run

Discrete Event Examples

Consider two nodes on an Ethernet

Simple queueing 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 reciving pkt
t=1.01: B's NIC concludes pkt
    B's NIC passes pkt to app
```

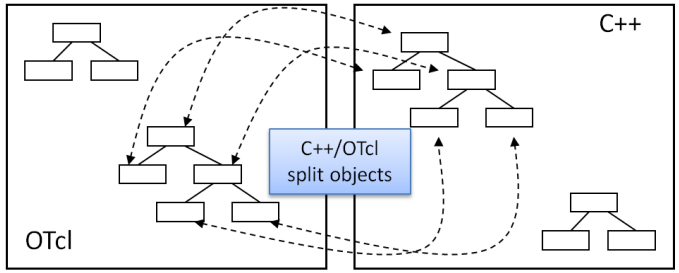
Ns Architecture

- Object-oriented (C++, OTcl)
- Modular approach
  - Fine-grained object composition
- Reusability
- Performance (speed and memory)
- Careful planning of modularity

C++ and OTcl Separation

- "data" / control separation
  - C++ for "data":
    - per packet processing, core of ns
    - fast to run, detailed, complete control
  - OTcl for control:
    - Simulation scenario configurations
    - Periodic or triggered action
    - Manipulating existing C++ objects
    - fast to write and change
- running vs. writing speed
- Learning and debugging (two languages)

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

Basic Tcl

Variables

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

Functions and expressions:

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

Control flow:

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

Procedures:

```
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.
- 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:
        Whats up, dude?
}

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

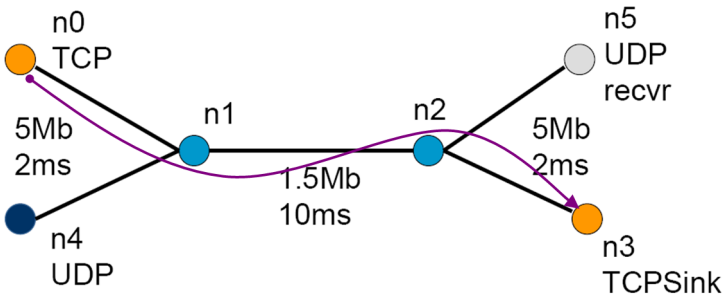
⇒ can easily make variations of existing objects (e.g., TCP, TCP/Reno)

Using ns-2

NS scripting

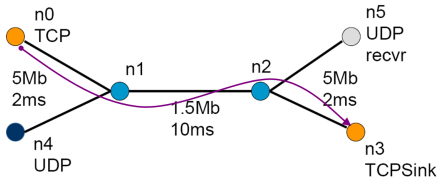
- Create the event scheduler
- Turn on tracing
- Create network
- *Setup routing*
- *Insert errors*
- Create transport connection
- Create traffic (applications)
- Transmit application-level data

An Example - TCP



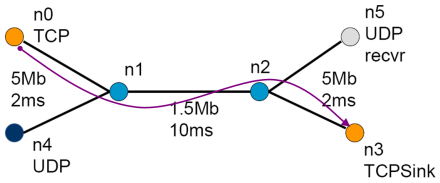
- Simple scenario with TCP and UDP connections

Step 1: Scheduler & Tracing



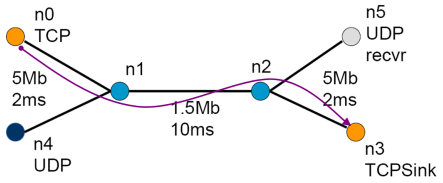

```
#Create scheduler
set ns [new Simulator]
#Turn on tracing so you have simulation logs
set f [open out.tr w]
$ns trace-all $f
set nf [open out.nam w]
$ns namtrace-all $nf
```

Step 2: Create Nodes



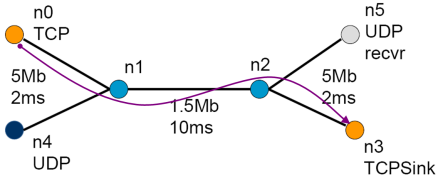

```
# Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
```


Step 3: Create Links/Queues



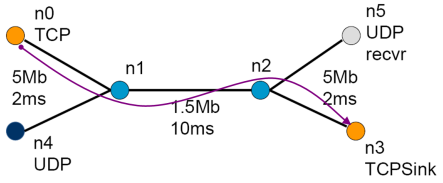

```
# Create links
$ns duplex-link $n0 $n1 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n2 $n3 5Mb 2ms DropTail
# Specify topology
$ns duplex-link-op $n0 $n1 orient right-down
$ns duplex-link-op $n1 $n2 orient right
...
# Queue limit optional
$ns queue-limit $n1 $n2 25
$ns queue-limit $n2 $n1 25
```


Step 4: Attach Agents



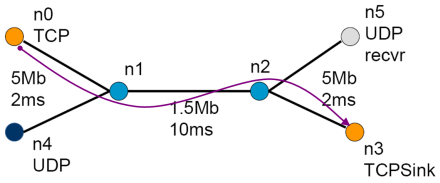

```
# Create TCP agents
set tcp [new Agent/TCP]
set sink [new Agent/TCPSink]
# Attach the agents to nodes
$ns attach-agent $n0 $tcp
$ns attach-agent $n3 $sink
# Set out connection between agents
$ns connect $tcp $sink
```


Step 5: Attach Applications




```
# Create and attach an application
set ftp [new Application/FTP]
$ftp attach-agent $tcp
# Start the application traffic at 1.1 sec
$ns at 1.1 "$ftp start"
```

Step 6: Stop, Close Trace and Finish




```
# Give stop time
$ns at 2.0 "finish"
# End of simulation wrapper
proc finish {} {
global ns f nf
close $f
close $nf
puts "Running nam..."
exec nam out.nam &
exit 0
}
$ns run
```

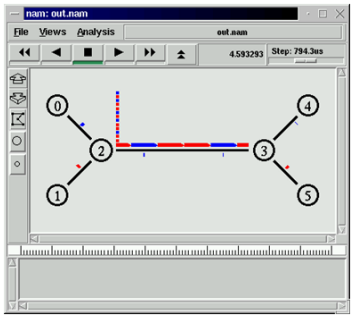

NAM: Visualization

- nam is used to visualize the simulation trace.

```
$ns namtrace-all [open test.nam w]
$ns namtrace-queue $n0 $n1
```

- Visualize trace in nam

```
Agent/TCP set nam_tracevar_ true
$tcp tracevar srtt_
$tcp tracevar cwnd_
```



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 $n2 $n3
```


Application-Level Simulation

- Features
  - Build on top of existing transport protocol
  - Transmit user data, e.g., HTTP header
- Two different solutions
  - TCP: Application/TcpApp
  - UDP: Agent/Message

NS-3 Key Features

- Trace output in ascii, or Pcap format
  - Use existing Pcap tools (e.g. ,Wireshark)
- Numerous trace points enabled via callbacks
- Python Bindings for most public functions
- Emulation mode
  - Integration with real networks/packets
  - Real-time scheduler
- Doxygen documentation
- Mercurial code repository
- Formal review/check-in procedure
- Quarterly releases

What's Next

- References
  - TELE302 Resources page (manual, tutorials, AWK)
- NS-2 Lab this week
  - Play with TCP / UDP traffic
  - Experiment with queue settings
  - Have fun with nam
- Coming next:
  - Queueing lectures
  - Assignment 1