IPAM Tutorial: Network Modeling and Traffic Analysis with ns-2

John Heidemann, USC/ISI
and Polly Huang, ETH-Zurich
14 March 2002

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 *comparision* 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

Alternatives

- experimentation → operational details, but
  - private laboratories
  - public testbeds (ex. CAIRN)
  - shared labs (ex. Utah Emulab)
  - → limited scale
  - → expensive, limited fexibility
  - → better, but higher overhead
- analysis → can provide understanding
  - but limited details
- other simulators → important niches
  - custom simulators
  - other general sims
  - → limited re-use
  - → fill niches

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: drop-tail, RED, fair queueing, etc.
- Physical media:
  – wired (point-to-point, LANs), wireless (multiple propagation models), satellite

ns status

- size: about 200k loc each C++ and Tcl, 350 page manual
- user-base: >1k institutions, >10k users
- platforms: basically all Unix and Windows
- releases about every 6 months, plus daily snapshots (next release: March 2002)

Outline

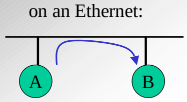
- **Concepts**
- Getting Started
- Fundamental tcl, otcl and ns
- Current ns activities
- Case Studies [Polly Huang]

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)

Discrete Event Examples

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

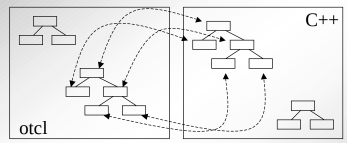
ns Software Structure: object orientation

- 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/documented 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.

Outline

- Concepts
- **Getting Started**
- Fundamental tcl, otcl and ns
- Current ns activities
- Case Studies

Installation

- http://www.isi.edu/nsnam/ns/
  - download ns-allinone
  - includes Tcl, OTcl, TclCL, ns, nam, etc.
- mailing list: ns-users@isi.edu
- documentation (on web at URL above)
  - Marc Gries tutorial
  - ns manual

Hello World

simple.tcl:

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

swallow 74% **ns simple.tcl**
Hello World!
swallow 75%

Hello World, Deconstructed

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 the simulator, executing events

Outlines

- Essentials
- Getting Started
- **Fundamental tcl, otcl and ns**
- Current ns activities
- Case Studies

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.

=> 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 {} {
    # instance variable
    # (state for the object)
    $self instvar age_
    puts "$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)

Basic ns-2

- Creating the event scheduler
- Creating network
- Computing routes
- Creating traffic
- Inserting errors
- Tracing

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 10Mb 100ms DropTail
  - specifies bandwidth, delay, and queue type
    - DropTail, RED, CBQ, FQ, SFQ, DRR

Routing Alternatives

- Unicast
  - $ns rtproto <type>
  - <type>: Static, Session, DV, cost, multi-path
- Multicast
  - $ns multicast
  - $ns mrtproto <type>
  - <type>: CtrMcast, DM, ST, BST
- (by default: static routing and no multicast)

Traffic in ns

- simple two layers: transport and application
- transports:
  - TCP, UDP, etc.
- applications:
  - web, ftp, telnet, etc.
  - may draw upon statistical models or traces

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 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 Traffic: Over 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: Over 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)

Controlling Object Parameters

- Almost all ns objects have *parameters*
  - ex. Application/Traffic/Exponential has *rate* and *packetSize*
  - set parameters in Tcl:
    - set etraf [new Application/Traffic/Exponential]
    - $etraf set rate_ 1Mb
    - $etraf set packetSize_ 1024
- Class selects object type, parameters control details

Inserting Errors

- Creating Error Module
  - set lossmod [new ErrorModel]
  - $lossmod set rate_ 0.01
  - $lossmod unit pkt
  - $lossmod ranvar [new RandomVariable/Uniform]
  - $lossmod drop-target [new Agent/Null]
- Insert Error Module into a Link
  - $ns lossmodel $loss_module $n0 $n1

Tracing

- 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>
    + 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
```

- Trace packets on all links in nam-1 format
  - $ns namtrace-all [open test.nam w]

Other topics

- multicast
- queueing alternatives
  - DropTail, RED, FQ, etc.
- more complex traffic models
  - web
  - traces
- import topology models
  - GT-ITM
  - inet
- wireless nodes
  - ad hoc routing, directed diffusion
  - radio propagation
  - mobility
  - node energy
- LANs
- satellite links

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 *parameters*
  - $tsrc set windowInit_ 4
- easy to change *whole implementations*
  - set tsrc2 [new Agent/TCP/NewReno]
  - set tsrc3 [new Agent/TCP/Vegas]

⇒ compare alternatives and experiment!

Outline

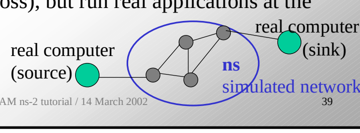
- Essentials
- Getting Started
- Fundamental tcl, otcl and ns
- **Current ns activities**
- Case Studies

Current ns Activities

- Hybrid simulation:
  - **Emulation**: mixing real and simulated nodes
  - **Abstract simulation**: comparing detailed and abstract simulations
  - **Approx-sim**: mixing packet-level sim and analysis
- Traffic modeling:
  - **just-in-time model generation**
  - **model scale-up**

Emulation

- Idea: combine simulation with real packets
- Applications:
  - simulate denial-of-service attacks to real hardware
  - simulate a large network (w/controlled cross traffic or loss), but run real applications at the edge



Abstract Simulation

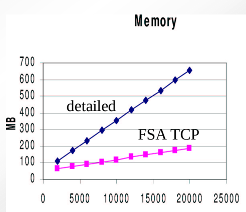
- Idea: support *multiple* levels of abstraction in one simulator
  - users select the right level of abstraction for their task
  - can *compare* detailed and abstract simulations to validate results
- Many kinds of abstraction:
  - routing, packet propagation, traffic model, etc.

Abstract Simulation Example: Finite-State TCP

- Idea:
  - Replace detailed TCP with finite-state machine model
  - possibly suitable for background traffic

=> greatly reduces memory requirements

- Validate against detailed models
- Huang & Heidemann, "Capturing TCP Burstiness in Lightweight Simulations", CNDS 2001



Abstraction Example 2: Analytic Pre-filtering

- Packet level simulators very accurate, but much slower than analytic approaches
- Often, large chunks of the simulation space are uninteresting
  - Obviously bad, or obviously good

⇒ *Use fast, approximate analytic techniques to pre-filter away uninteresting scenarios*

  - Do detailed simulations for the interesting scenarios

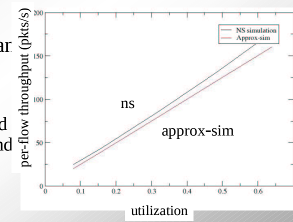
Pre-filtering: Approach

Approximate hybrid queuing:

- use TCP model [Padhye98a] for *long flows*:
  $T = k/(RTT*\mathrm{sqrt}(p))$
- model *short flows* as fluid (using new, simple approx)
  $T = n\,\lambda / (1-p)$
- iteratively solve to find stable *fixed-point*
  - queueing solution to (delay, drop) from (througput)
  - short and long TCP models to solve (throughput) given (delay, drop)
- handle both drop-tail and RED queueing policies

Pre-filtering: Status

- proof that algorithm will converge
- good convergence times (>10x faster than packet-level sim)
- reasonable accuracy
  - examined for long and short flows, simple and more complex topologies
- details: Dutta, Goel, Heidemann, ISI TR-550, Nov 2001



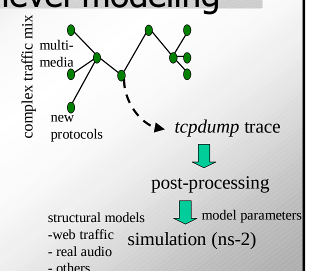
Just-in-time Model Parametrization

- Appropriate traffic models are essential to serve as the input of simulation
  - replaying traces doesn't capture congestion reactivity
- Traffic isn't all the same:
  - frequently and unpredictably changes
    - temporal-variations in web traffic
    - new traffic types: Napster, p2p file sharing
  - different at different places

⇒ No single model and parameters can fit them all; manual model parameterization is too slow

Our approach: Trace-driven, application-level modeling

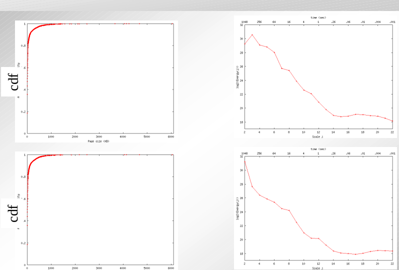
- take *tcpdump* packet trace
- process trace to get parameters for *structural model* of traffic
  - processing is O(minutes)
- feed model into simulator



Model validation

ISI trace
08/13/2001
14:00–15:00

rapidly-generated model



First-order statistics (page size) Wavelet scaling plot

Outline

- Essentials
- Getting Started
- Fundamental tcl, otcl and ns
- Current ns activities
- **Case Studies**

Model Scale-up

- Problem: want to test on *tomorrow's* routers (10GB/s), but lack traces
- Approach: synthesize a traffic model from lower-speed traces, scale it up
  - can vary bandwidth, users ~ *k* bw
  - evaluating simulation scale-up vs. alternatives such as trace scaling or trace merging