Internet Programming & Protocols
Lecture 15

Emulation

Simulation

www.cs.utk.edu/~dunigan/ipp/

Evaluating the performance of TCP

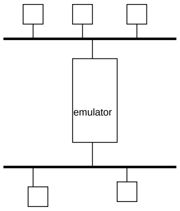
- Experimental
  - Standalone testbeds
  - Emulator testbeds
  - Live tests on the Internet
  - Active tools (iperf, ping, traceroute) / passive tools (tcpdump/netflows)
  - Collect flow packet trace, full traffic traces
  - Instrumented kernels (Web100)
- Theoretical
  - Analytical models to characterize a TCP flow
  - Stochastic/statistical models to characterize flow interactions (background)
  - Queuing models to characterize router behavior
  - Linear feedback (control) systems to characterize optimal solutions
- Simulation
  - Repeatable, flexible, instrumented

Real tests or simulations

- **Live internet tests**
  - See results in ultimate environment
  - Real TCP stacks/OS, traffic
  - Vary time and host/paths
  - Worry about impact?
- **Test beds**
  - Controlled traffic, but real OS
  - Usually LAN based, no queuing
  - Repeatable
  - Not very good for cross-traffic
- **Emulators**
  - Same as testbed
  - Plus control delay, loss, data rates, dup's, out-of-order
  - Easy to reconfigure
- **Simulations**
  - Easily reconfigured
    - Complex topology
    - Vary TCP flavor
  - Repeatable
  - Detailed feedback/instrumentation
  - Add delay, loss, cross-traffic, queues
  - Randomness for confidence
  - Investigate "new" networks/protocols
  - cheap
  - Can be slow
  - Not real TCP
- Need tools to probe and measure

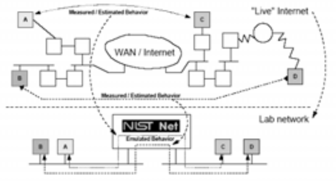
Network emulation

- Real OS network stack and application testing in a controlled testbed
  - Can use all your network tools, ping, iperf, tcpdump, ssh, nfs ….
- Use a "modified" UNIX box as a router (2 NICs) that can introduce
  - Packet loss
  - Packet delay (select your RTT)
  - Packet reordering
  - Bandwidth limits (select bandwidth)
  - Different queuing disciplines
  - Src.port dst.port filters
- Freeware implementations
  - NISTNet, netem (linux)
  - dummynet (freebsd)
- Use to evaluate
  - Your network application protocol
  - Your cool mods to the kernel's TCP stack
  - OS's TCP behavior



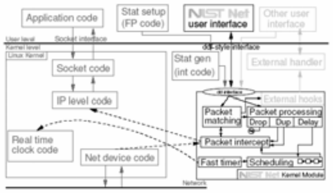
NISTNet

- Linux kernel module plus configuration utility
- Network in a box
- Emulates packet loss (random and congestion-dependent), reordering, bandwidth limits, delay (fixed and variable), duplicates, queues



NISTNet implementation

- Kernel module intercepts all IP packets
- Introduces delay/loss etc. based on source/dest filters
- Module has packet queues, timer, filter tables
  - Uses MC146818 realtime clock (RTC) for timer interrupts (122 us)
- Configuration utility (application) configures module filters

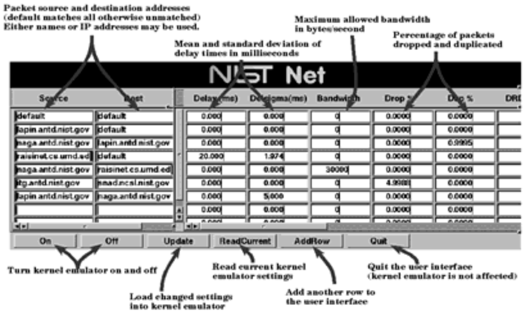


NISTNet configuration

- Command (script) based or GUI
- Command line options (cnistnet)
  - **-a src[:port[.protocol]] dest[:port[.prot]] [cos]**
    - **add new**
      [--delay delay [delsigma[/delcorr]]]
      [--drop drop_percentage[/drop_correlation]]
      [--dup dup_percentage[/dup_correlation]]
      [--bandwidth bandwidth]
      [--drd drdmin drdmax [drdcongest]]

```
cnistnet u   (start nistnet

cnistnet –R –G  (status report on filters)

cnistnet -a hosta pinto10 --drop 10.0006 --bandwidth 10000000 –delay 50

cnistnet -a pinto10 hosta --bandwidth 10000000 –delay 50

Cnistnet –a 0 pinto10:5001 –drop 0.01 –delay 120
```

NISTNet GUI



Packet source and destination addresses
(default matches all otherwise unmatched)
Either names or IP addresses may be used.

Mean and standard deviation of
delay times in milliseconds

Maximum allowed bandwidth
in bytes/second

Percentage of packets
dropped and duplicated

Source	Dest	Delay(ms)	Delsigma(ms)	Bandwidth	Drop %	Dup %
default	default	0.000	0.000	0	0.0000	0.0000
lapin.antd.nist.gov	default	0.000	0.000	0	0.0000	0.0000
naga.antd.nist.gov	lapin.antd.nist.gov	0.000	0.000	0	0.0000	0.9995
raisinet.cs.umd.ed	default	20.000	1.974	0	0.0000	0.0000
naga.antd.nist.gov	raisinet.cs.umd.ed	0.000	0.000	30000	0.0000	0.0000
itg.antd.nist.gov	nmad.ncsl.nist.gov	0.000	0.000	0	4.9988	0.0000
lapin.antd.nist.gov	naga.antd.nist.gov	0.000	5.000	0	0.0000	0.0000
		0.000	0.000	0	0.0000	0.0000

Turn kernel emulator on and off

Load changed settings
into kernel emulator

Read current kernel
emulator settings

Add another row to
the user interface

Quit the user interface
(kernel emulator is not affected)

NISTNet packet loss

- Random or congestion based packet loss
- --drop drop_percentage[/drop_correlation]
- Supports a RED-like queue with min and max thresholds
  - --drd drdmin drdmax [drdcongest]
- Option for ECN notification

NISTNet delay models

- Packet delay can be fixed or random
- For random delays user can specify
  - Mean
  - Standard deviation
  - Linear correlation
  - Default: derived heavy-tail distribution

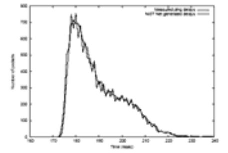
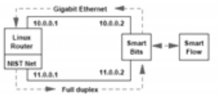
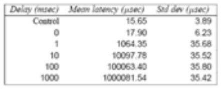


Figure 6: Real vs. NIST Net-synthesized delay distributions

NISTNet overhead

- NISTNet adds some overhead for a packet passing through the "router"
- Calibration with packet generator
- 2 us overhead
- Additional variance from granularity of RTC interrupts (122 us)
- OK on gigE NICs
  - Though clumping above 100 mbs
  - 20,000 packets/sec

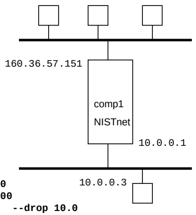




Delay (msec)	*Mean latency (μsec)*	*Std dev (μsec)*
Control	15.65	3.89
0	17.90	6.23
1	1064.35	35.68
10	10097.78	35.52
100	100063.40	35.80
1000	1000081.54	35.42

NISTnet example

- comp1.cs.utk.edu has 2 NICs and NISTnet module installed with IP forwarding
- 2nd interface is 10.0.0.1
- Target box is 10.0.0.3
  - Other CS boxes need static route to 10.0.0.3 via comp1
- Examples
  - ping from whisper
  - ping from cetus1
  - iperf from cetus2



```
cnistnet -u
cnistnet -a whisper.cs.utk.edu 10.0.0.3 --delay 50.000
cnistnet -a 10.0.0.3 whisper.cs.utk.edu  --delay 50.000
cnistnet -a cetus1.cs.utk.edu 10.0.0.3 --delay 40.000  --drop 10.0
cnistnet -a cetus2.cs.utk.edu 10.0.0.3 --delay 20.000  --bandwidth 500000
```

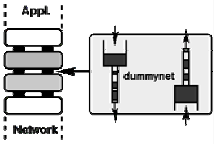
Linux netem

- Part of 2.4 and 2.6 kernel
- Supports packet delay, loss, duplication, reordering (tc command)
  - tc qdisc add dev eth0 root netem delay 100ms
  - tc qdisc change dev eth0 root netem loss .1%
  - tc qdisc change dev eth0 root netem duplicate 1%
  - tc qdisc change dev eth0 root netem gap 5 delay 10ms
- Rate limits are provided by existing Linux queuing services
  - tc qdisc add dev eth0 root handle 1: prio
  - tc qdisc add dev eth0 parent 1:3 handle 30: netem delay 200ms 10ms distribution normal
  - tc qdisc add dev eth0 parent 30:1 tbf rate 20kbit buffer 1600 limit 3000
  - tc filter add dev eth0 protocol ip parent 1:0 prio 3 u32 match ip dst 65.172.181.4/32 flowid 10:3
- Filtering on net/host/ports
  - tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 match ip src 4.3.2.1/32 match ip sport 80 0xffff flowid 10:1

---

dummynet

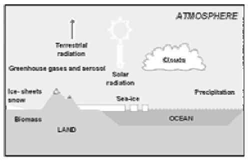
- Part of FreeBSD
- Managed with ipfw
- Provides delays, queues (WFQ), packet loss, bandwidth limits, multipath
- Filter on src/dst port or port range



```
  ADSL path to moon
ipfw add pipe 3 ip from any to any out
ipfw add pipe 4 ip from any to any in
ipfw pipe 3 config bw 128Kbit/s queue 10 delay 1000ms plr 0.01
ipfw pipe 4 config bw 640Kbit/s queue 30 delay 1000ms
```

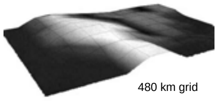
Simulation

- The three branches of science
  - Theory
  - Experiment
  - Simulation
- Computer simulation a cornerstone of today's scientific research
  - Weather forecasting (hurricane path prediction)
  - Global climate modeling
  - Vehicle design (crash test simulation)
  - Assembly line simulation
  - Super nova simulation
- Simulators for training/education
  - Flight simulators
  - Physics "experiments"

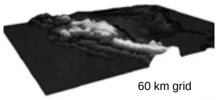


Continuous vs discrete simulations

- Continuous simulation – simulation time moves in monotonic increments
  - Climate/weather modeling
  - Game of life
  - Real time required is a function of computation required at each time step and speed (and number) of computers
    - Can be faster than realtime –weather forecasting
- Discrete event simulation
  - Network simulation
  - Simulation time moves in jumps based on time of "next event"
    - E.g., packet arrives in 2 seconds, move simulation clock ahead by 2 seconds
  - Real time required is a function of number of events -- lots of nodes, high packet rates will take much longer than real time



480 km grid

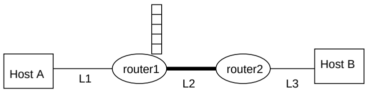


60 km grid

How do initial conditions affect result?

Does butterfly flapping its wings in Brazil affect result?

Simulating a network

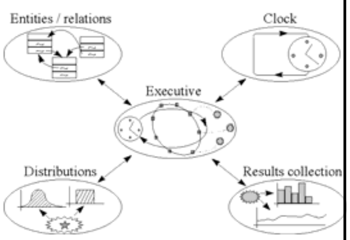


- Define a topology
- Define component characteristics
  - Router: queuing discipline (FIFO), queue size
  - Link: delay, bandwidth, bit error rate (BER) loss probability
  - End nodes: TCP flavor, window size, del ACK, MSS, timer tick resolution
    - Packet source (infinite (FTP), telnet, http, constant bit rate)
- Simulation is based on discrete events – packets moving from one component to the next

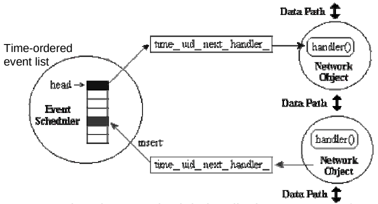
---

Discrete event simulation

- Permanet entities
  - Nodes, routers, links
- Transient entities
  - Packets
  - Timer events
- Software handler for each permanent entity
- Executive (scheduler) drives the simulation based on a time-ordered event list
- Components for random numbers and statistical distributions
- Instrumentation for tracing events and reporting results

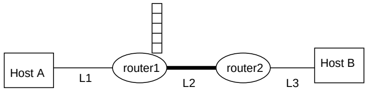


Event scheduling



- Executive (event scheduler) pulls the next event from the event list and advances the simulation clock to the time and invokes the indicated handler
  - – E.g., pass packet to “link handler”, link handler will insert new event on list at time = now + link_delay
  - – Or TCP handler may add an event for “packet timeout” at time = now + RTO (when ACK arrives for that packet later, it may remove the event from the event list)

Life of a simulated packet



- Host A’s TCP passes packet to link_handler for L1
- Link_handler puts event on list for time now+transmission delay + link delay
- Scheduler advances simulation clock and invokes router_handler for router1
- Router handler puts event on list at time based on queuing delay (number in queue) or drops!
- Scheduler advances clock, invokes link_handler for L2
- L2 link_handler puts event on list for time = now +delays
- Scheduler advances clock, invokes router_handler for router2
- Router_handler puts event on list based on queueing delay
- Scheduler advances clock, invokes link_handler for L3
- L3 link handler puts event on list based on delays
- Scheduler advances clock invokes Host B TCP, which sends back an ACK …

Simulation software

- Write your own simulation in C or Java
  - – Need to write scheduler and all the handlers (one for each flavor of TCP…)
- Simulation languages
  - – SIMSCRIPT, SIMULA
  - – Generic simulation framework provided
    - Scheduler, tracing, GUI, graphing, statistical packages
  - – Still have to build components of the “system” you are simulating
- Pre-built simulators for the “system” you are interested in
  - – Network simulation: ns, OPNET, SSFnet
  - – Ease of use, GUI, debugging, tracing, speed, cost
  - – Features: links, TCP’s, UDP, routing, wireless, link layer, scalable
  - – Reliability (e.g., trusted/realistic implementation of SACK)
  - – Extensible – modify or add “new” protocols

ns network simulator

- Discrete event simulator (free ☺)
- Packet-level
- Link layer and up
- Wired and wireless
- History
  - – Columbia NEST
  - – UCB REAL
  - – ns-1
  - – ns-2
    - 100K lines of C++
    - 70K lines of OTcl
    - 30K lines of test suite
    - 20K lines of documentation
- Platforms: UNIX boxes, some pieces on Windows (ns, nam)

Functionality of ns

- Wired world
  - – Point-to-point link, LAN
  - – Unicast/multicast routing
  - – Transport
    - UDP
    - TCP (Tahoe, Reno, NewReno, SACK, FACK, HSTCP ….)
  - – Application layer
- Wireless
  - – Mobile IP
  - – Ad hoc routing
- Tracing, visualization, animation, various utilities

Object-Oriented

+ Reusability

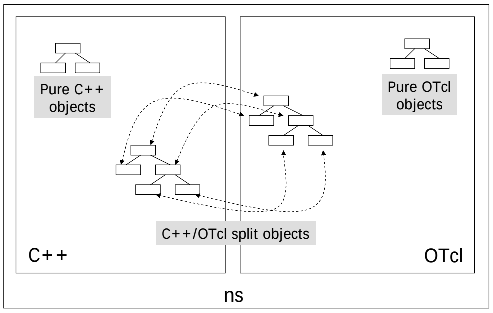
+ Maintenance

– Performance (speed and memory)

– Careful planning of modularity

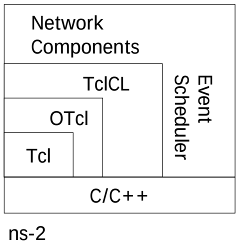
- Combination (ugly) of C++ and Tcl
  - – C++ for “data”
    - Per packet action
      - – Fast: event scheduler, TCP flavors
    - You only mess with this if you’re extending the simulator
  - – OTcl for control
    - Periodic or triggered action
    - Tcl is what you’ll be using
  - + Compromise between composibility and speed
  - – Learning and debugging

OTcl and C++: The Duality



Extending Tcl Interpreter

- OTcl: object-oriented Tcl
- TclCL: C++ and OTcl linkage
- Discrete event scheduler
- Data network components
  - Link layer and up
  - Emulation support



Hello World - Interactive Mode

```
swallow 71% ns
% set ns [new Simulator]
_o3
% $ns at 1 "puts \"Hello World!\""
1
% $ns at 1.5 "exit"
2
% $ns run
Hello World!
swallow 72%
```

Hello World - Batch Mode

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%
```

To run ns on the CS lab machines (cetus/hydra) you'll need to add some stuff to your PATH and LD_LIBRARY. See the README in ~dunigan/ipp05/ns

Sample scripts are there as well

Intro to Tcl

- Interpreted command language
- Tcl script consists of one or more commands separated by new lines or semicolons
- Command is followed by 0 or more words or arguments separated by tabs or white space. State information is stored in variables

  set a 5
  set b [expr $a +6] ; # set b = a+6 is NOT what you want
  puts "b is $b"
- Use $ to retrieve value of variables
- Use # for comments (;# at end of line)
- […] evaluates the command inside the [ ] and returns the value
- " .. " is a string, $variables value are substituted
- { … } defers evaluation
- Normal C operators and precedence + - * / | & && || == > < !=

More Tcl

```
# conditionals and looping
if {$a < 17 } {
        set x [expr $a/(3-$z)]
}  else {
        incr x
}
while {$bob == $alice} { … }

for {set i 0} {$i < 10} {incr i 3} {
        puts "vector element $i: $vector($i)"
}
#  also has break and continue like C
```

More Tcl

```
# lists
set x {1 3 a}
set y [lindex $x 1] ; # y is 3
set length [llength $x]
foreach val $x {  puts “val is $val” }
set delay [lindex $argv 1]  ;# command line args

if {$argc > 1} {
    set proto  [lindex $argv 0]
    set buffer [lindex $argv 1]
    set lrate  [lindex $argv 2]
} else {
    puts "usage: ns test.tcl <protocol> <buffer> <error rate>"
}

# Tcl script to echo command line arguments
puts "Program: $argv0"
puts "Number of arguments: $argc"
set i 0
foreach arg $argv {
        puts "Arg $i: $arg"
        incr i
}
```

More Tcl

```
# strings
set name “Bob and Alice”
set lth [string length $name]
if {$x == “test”} {
        append x “ing”
        set output [format “%.1f“ $rate]
}

# i/o
set trace_wnd [open out.wnd w]
puts $trace_wnd "$now $curr_wnd“
close $trace_wnd
```

More Tcl

```
# procedures  “new” commands
#  need global to reference external variables
set a 43
set b 27
set bob “Bob”

proc test { a b } {
        global bob
        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, $k)]”
                } else {
                        puts “$bob k >= 5, mod = [expr $d % $k]”
                }
        }
}

test 43 27

# usual builtin math functions sqrt(), sin(),pow(), log()…
```

Basic OTcl

```
Class Mom
Mom instproc greet {} {
    $self instvar age_
    puts “$age_ years old mom: How are you
    doing?”
}

Class Kid -superclass Mom
Kid instproc greet {} {
    $self instvar age_
    puts “$age_ years old kid: What’s up,
    dude?”
}
```

```
set mom [new Mom]

$mom set age_ 45

set kid [new Kid]

$kid set age_ 15



$mom greet

$kid greet
```

ns has several new “classes”, public variables, methods

set ns [new Simulator]

set n1 [$ns node]

$ns at “10.0” finish

set tcp [new Agent/TCP/Sack1]

$ns attach-agent $n0 $tcp

set curcwnd [$tcp set cwnd_]

Next time …

- More ns

assignment 7