Network Modeling and Traffic Analysis with ns-2

CS 2520/TELCOM 2321
High Speed WANs
Fall 2004

October 21, 2004

Slides loosely based on tutorials by Polly Huang (ETH), John Heidemann (USC/ICSI) and Bianca (CMU).

Outline

- *NS in general*
- Basic knowledge to work with ns
  - Tcl and OTcl
- Getting start
- Demo
- Environment setup

NS-2 = network simulator

- A *discrete event* simulator
- 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

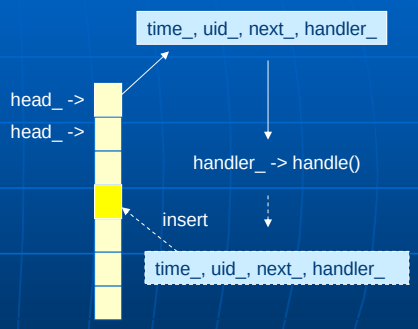
- Support network research and education
  - Protocol design, traffic studies, etc.
  - Protocol comparison
- Provide a collaborative environment
  - Freely distributed, open source
  - Allow easy comparison of similar protocols
  - Increase confidence in results
    - Experts develop models
    - Many researchers use and prove the models

NS is good for

- Evaluate performance of existing network protocols.
- Prototype and evaluate of new protocols
- Simulate large-scale simulations, which is not possible in real experiments

NS – How it work

- 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: single thread of control ➔ no locking or race conditions



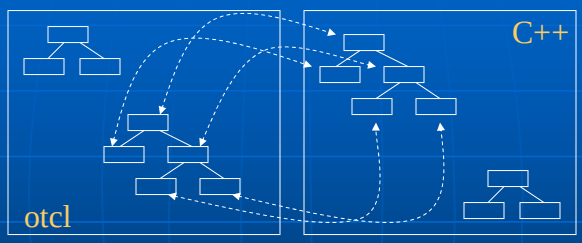
Ns – Models

- Traffic models and applications:
  - Web, FTP, telnet, cbr, real audio
- Transport protocols:
  - Unicast: TCP(Reno, Vegas, etc.), UDP
  - Multicast: SRM
- Routing and queuing:
  - Wired routing, ad hoc routing and directed diffusion
  - Queuing protocols: drop-tail, RED, fair queuing, etc.
- Physical media:
  - Wired (point-to-point, LANs), wireless (multiple propagation models), satellite

Ns software structure

- Two languages
  - *C++* for packet processing
    - *Fast to run*, detailed, completed 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

otcl and C++: The Duality



- Otcl (object variant of Tcl) and C++ share class hierachy

Outline

- NS in general
- *Basic knowledge to work with ns*
  - *Tcl and OTcl*
- Getting start
- Demo
- Environment setup

Basic tcl

Variables:
```
set x 10
puts “x is $x”
```

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

Procedure (function):
```
proc pow { x n } {
    if {$n == 1}{return $x}
    set part [pow x [expr $n-1]]
    return [expr $x*$part]
}
```

Control flow:
```
if {$x>0} {return $x} else {
        return [expr -$x]}
while {$x > 0} {
        put $x
        incr x-1
}
```

- $ for de-referencing
- Spaces - important
- {} defines a block
- set, puts
- proc definition:
  proc name args body

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 a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15
$a greet
$b greet
```

- instead of single class declaration
  → **multiple definitions**
- **instproc** adds class methods
- **instvar** adds instance variable, and brings them to the local scope
- **$self** : **this** in Java, C++
- all methods **virtual** (as in Java)

Ns – Hello World

Simple.tcl

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

$ns run
```

oxygen74% **ns simple.tcl**
Hello World!
oxygen75%

- Create a simulator, put in var ns
- Schedule an event at time t=1to print Hello World
- Exit at time t=1.5
- Run the simulator, executing events

Outline

- NS in general
- Basic knowledge to work with ns
  - Tcl and OTcl
- *Getting start*
- Demo
- Environment setup

Basic-ns2

- Creating the event scheduler
- Creating network topology
- Computing routing scheme
- Creating traffic
- 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

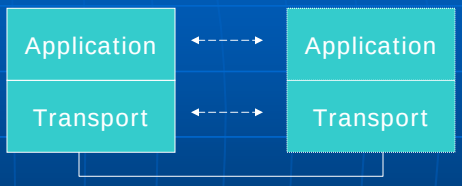
- Nodes
  - set n0 [$ns node]
  - set n1 [$ns node]
- Links & Queuing
  - $ns duplex-link $n0 $n1 10Mb 100ms DropTail
  - Specifies, bandwidth, delay and queue type
    - Queue types: Droptail, RED, CBQ, FQ, SFQ, DRR

Routing Alternatives

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

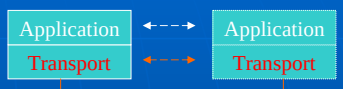
Traffic in ns

- Simple two layers: transport and application
- Transports:
  - TCP, UDP, etc.
- Applications:
  - Web, ftp, telnet, etc.
  - May draw upon statistical models or traces

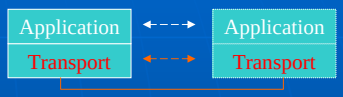


Create connection: TCP

- 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

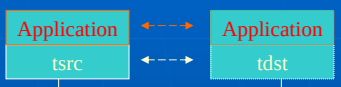


Create connection: UDP



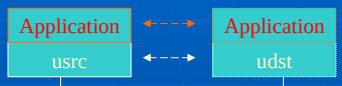
- Source and sink
  - Set usrc [new Agent/*UDP*]
  - Set udst [new Agent/*NULL*]
- Connect to nodes and each other
  - $ns attach-agent $n0 $usrc
  - $ns attach-agent $n1 $udst
  - $ns connect $usrc $udst

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 cbr [new Application/*Traffic/CBR*]
- Exponential or Pareto on-off
  - set exsrc [new Application/*Traffic/Exponential*]
  - set exsrc [new Application/*Traffic/Pareto*]

Tuning parameters

- Almost all ns objects have parameters
  - Traffic generator: exponential has rate and packet size
  - Set parameters in Tcl:
    - set etraf [new Application/Traffic/Exponential]
    - $etraf set rate_ 1 Mb
    - $etraf set packetSize_ 1024
  - Class selects object type, parameters control detail

Tracing

- Trace packets on all links into test.out
  - $ns trace-all [open test.out w]
- Trace Format

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id

```
r : receive  (at to_node)                  src_addr : node.port (3.0)
+ : enqueue  (at queue)                    dst_addr : node.port (0.0)
- : dequeue  (at queue)
d : drop     (at queue)
```

```
r 1.3556 3 2 ack 40 ------- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ------- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ------- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ------- 2 1.0 3.1 157 207
```

- *Trace all – simple, but not always good!*

More Tracing

- Tracing specific links
  - $ns trace-queue $n0 $n1 $f
- Tracing variables
  - set cwnd_chan_ [open all.cwnd w]
  - $tcp trace cwnd_
  - $tcp attach $cwnd_chan_

Ns2 – general procedure

- Create simulator instance
- Create network (nodes and links)
  - Specify bandwidth, delay, queue size, etc.
- Create traffic (tcp, udp, cbr, ftp, etc.)
  - Create agents and connect them
  - Modify parameters, if need to
- Tracing
  - many levels of granularities: all, per flow, per queue, etc.

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

Example (from ns by example)

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

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

#Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#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
```

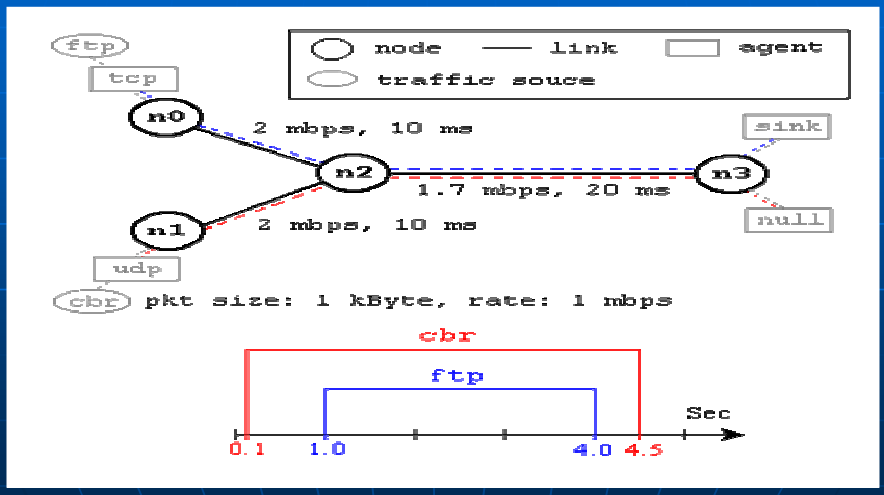
```
#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

#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 the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam out.nam &
        exit 0
}
```



```
#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 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
#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
```

```
#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"

#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
```

Example (continue)



Demo

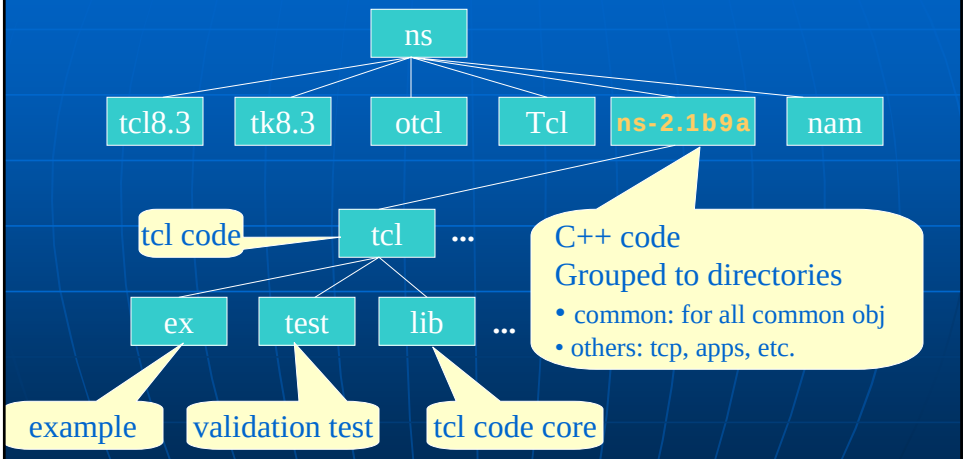
Compare to Real World

- 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 implementation
  - Set tsrc2 [new Agent/TCP/NewReno]
  - Set tsrc3 [new Agent/TCP/Vegas]

Outline

- NS in general
- Basic knowledge to work with ns
  - Tcl and OTcl
- Getting start
- Demo
- *Environment setup*

Explore ns source code



Environment setup

- Only using existing protocols (only run tcl script)
  - Set *path* to ns
  - Set *library path* (LD_LIBRARY_PATH and TCL_LIBRARY)
- Modify or add a protocol
  - link source tree, include your c++ or otcl codes
  - Modify makefile
  - Build your own version of ns
  - Set path to your ns
  - Set library path to the same location above

Environment setup (cont.)

- CS
  - Log into any linux machines in department (oxygen, cooper, etc.)
  - Run this command
    - source /usr/local/contrib/ns/setns
  - This script should setup path and library path for ns for you
  - Root of ns: /usr/local/contrib/ns

(Note: I installed ns on oxygen, if you use the machine that has different architecture, it may not work)

Environment setup (cont.)

- Telcom
  - Log into any unix machines in department (paradox from outside telcom domain or other unix machines in the 8th floor lab)
  - Run this command
    - source /home/tele2321/setns
  - This script should setup path and library path for ns for you
  - Root of ns: /home/tele2321/ns

Question???