

Part 3: Network Simulator – 2

24.11.2004

1

NS2: Contents

- NS2 – Introduction to NS2 simulator

- Background info

- Main concepts, basics of Tcl and Otcl
 - NS2 simulation building blocks

- Some NS2 examples
- NS2 project work instructions

24.11.2004

2

References

Material based on following sources:

1. Haobo Yu, Nader Salehi, "NS2 tutorial", IEC'2000 ns workshop, San Diego, USA, June 2000.
2. John Heideman, "IPAM tutorial: Network modeling and traffic analysis with ns-2", presentation at the UCLA/Institute for Pure and Applied Mathematics, Los Angeles, USA, March 2002.

Both available from

- <http://www.isi.edu/nsnam/ns/ns-tutorial/index.html>

24.11.2004

3

What is NS2?

- Short characterization
 - discrete event network simulator
 - packet-level
 - link layer and up
 - wired and wireless
- A collaborative simulation platform
 - freely distributed, open source
 - developed by researchers in universities and research institutes
 - provide common reference \Rightarrow promote sharing
 - test suites \Rightarrow increase confidence in results
- Intended users
 - researchers
 - developers
 - educators

24.11.2004

4

History and status

- Brief history
 - REAL simulator by UCB (1989)
 - ns1 (Floyd and McCanne, then at LBL)
 - ns2
 - VINT project (Virtual InterNet Testbed)
 - LBL, PARC, UCB, USC/ISI
 - currently maintained at USC/ISI, with input from K. Fall, S. Floyd et al.
- Status
 - size: > 200k loc (lines of code) of C++ and Tcl, 350 page manual
 - user base: >1k institutions, >10k users
 - platforms: (almost) all Unix and Windows
 - Windows needs some manual work, Unix (Linux) is the preferred platform
 - releases about every 6 months, plus daily snapshots of the CVS archive
 - current version ns-2.27b9a, released Jan '04

24.11.2004

5

NS components

- ns, the simulator itself
- nam, the Network AniMator
 - for visualizing ns output
 - GUI for simple ns scenarios
- Pre-processing
 - traffic and topology generators
- Post-processing
 - simple trace analysis
 - using Awk, Perl, or Tcl

24.11.2004

6

NS models

- Traffic models and applications
 - web, FTP, telnet, constant bit rate, on-off
- Transport protocols
 - unicast: TCP (Tahoe, Reno, Vegas, ...), UDP
 - multicast: SRM
- Routing and queuing
 - wired routing (unicast, multicast), ad hoc routing, Mobile IP
 - queuing models: drop tail, RED, fair queuing
- Physical media
 - wired (point-to-point, LANs), wireless (multiple propagation models), satellite

24.11.2004

7

Installation

- <http://www.isi.edu/nsnam/ns/>
 - for easy installation, download ns-allinone
 - includes Tcl, Otcl, TclCL, ns, nam, etc.
 - to optimize size, it is possible to compile from pieces (see URL for details)
- Mailing list: ns-users@isi.edu
 - “subscribe ns-users” in body
 - for archive of mails see URL
- Documentation (on web at URL above)
 - Marc Greis tutorial
 - ns manual

24.11.2004

8

NS2: Contents

- NS2 – Introduction to NS2 simulator

- Background info

- Main concepts, basics of Tcl and Otcl

- NS2 simulation building blocks

- Some NS2 examples

- NS2 project work instructions

24.11.2004

9

NS architecture (1)

- Object-oriented & modular

- pros: code reuse (e.g., TCP variants), maintenance
 - cons: performance (speed and memory), careful planning of modularity

- Software structure

- uses two languages: C++ and OTcl (Object TCL)
 - to achieve separation of control- and packet level
 - C++ for packet processing
 - fast execution, detailed, full control over execution
 - to make simulator scalable, packet processing must be done at C++ level
 - OTcl for control
 - simulation setup, configuration, occasional actions (e.g., creating new TCP flows)
 - compromise between speed and abstraction level(s) offered to the user
 - draw back: need to learn two languages and debug in two “worlds”

24.11.2004

10

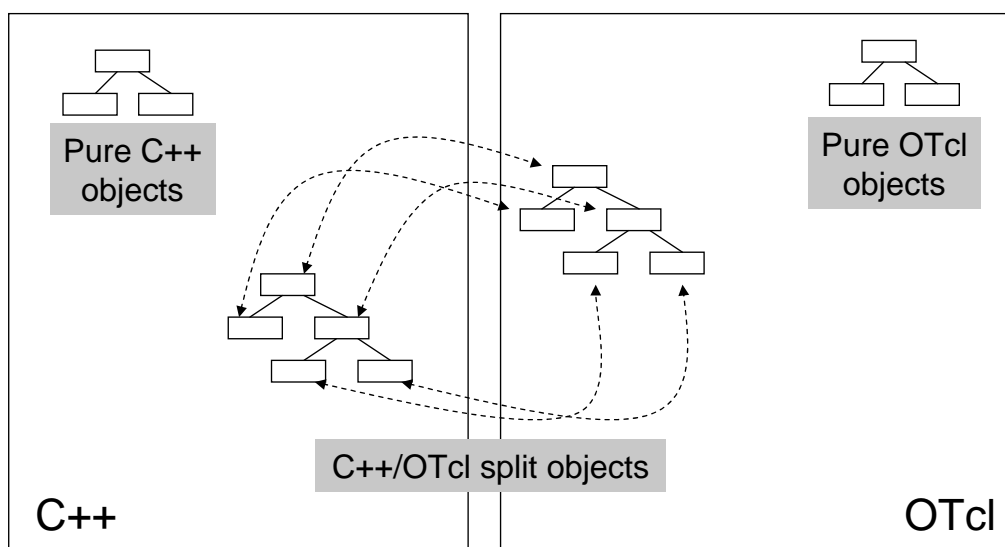
NS architecture (2)

- Architecture aims at scalability and easy extensibility
- Scalability
 - per packet actions need to be implemented such that execution is quick
 - achieved by separating control and packet handling
- Extensibility
 - must be “easy” for users to add own objects and functionality
 - fine-grained object composition:
 - basically, easy to understand role of each object and to identify which object(s) to modify
 - split C++/OTcl objects:
 - do not have to change anything at C++ level if new functionality only needed at OTcl level

24.11.2004

11

OTcl and C++: the duality



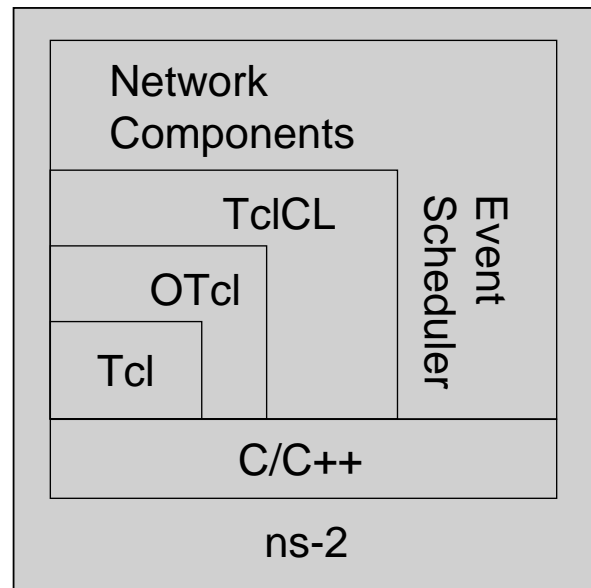
- OTcl and C++ share class hierarchy
- TclCL-library implements mechanisms that make sharing of functions, variables, etc., possible between C++ code and OTcl

24.11.2004

12

Software architecture

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



24.11.2004

13

Hello World!

```
simple.tcl
# Create the simulator object and assign it name "ns"
set ns [new Simulator]

# Schedule event at time 1 to print Hello World!
$ns at 1 "puts \"Hello World!\""

# ... and exit at time 1.5
$ns at 1.5 "exit"

# Run the simulation
$ns run

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

24.11.2004

14

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
}

for {set i 0} {$i < 10} {incr i}
    {puts $i}
```

Procedures:

```
proc fact {n} {
    if {$n == 1} {
        return 1
    } else {
        expr $n*[fact [incr n -1]]
    }
}
```

```
proc sum {} {
    global a b
    expr $a+$b
}
```

Tcl benefits:

- Tcl also contains lists, associative arrays, etc.
- Can use a real programming language to construct topologies, traffic sources, applications, etc.

24.11.2004

15

Basic OTcl

```
Class Person
# constructor
Person instproc init {age} {
    $self instvar age_
    set age_ $age
}

# method greet
Person instproc greet {} {
    $self instvar age_
    puts "$age_ years old: How
are you doing?"
}
```

```
Class Kid -superclass Person
# new greet-method
Kid instproc greet {} {
    $self instvar age_
    puts "$age_ years old kid:
What's up, dude?"
}

set person [new Person 45]
set kid [new Kid 15]

$person greet
$kid greet
```

⇒ Can easily make variations of existing things (e.g., TCP variants)

24.11.2004

16

NS2: Contents

- NS2 – Introduction to NS2 simulator
 - Background info
 - Main concepts, basics of Tcl and Otcl
 - NS2 simulation building blocks

- Some NS2 examples
- NS2 project work instructions

24.11.2004

17

Elements of ns2

- Assumption:
 - we only consider wired simulations (without routing)
- Important elements:
 - Create the event scheduler and random number generator
 - Create network
 - nodes and links
 - Create transport connection
 - TCP, UDP
 - Create applications
 - CBR, FTP
 - Setup tracing
 - trace queues and flows

24.11.2004

18

Creating the event scheduler

- Create event scheduler
 - set ns [new Simulator]
- Schedule events
 - \$ns at <time> <event>
 - <event>: any legitimate ns/tcl commands
- Start scheduler
 - \$ns run

24.11.2004

19

Creating random number generators

- Creating a pseudo random number generator (with heuristic seed)
 - set rng [new RNG]
 - \$rng seed 0
- Generating rv's from other distributions
 - using the class RNG
 - uniform rv's: \$rng uniform a b, \$rng integer k
 - exponential (with average 1): \$rng exponential
 - using the class RandomVariable
 - available distributions: uniform, exponential, hyper-exponential, Pareto
 - example: hyper-exponential

```
# Create and configure generator
set hypexp [new RandomVariable/HyperExponential]
$hypexp set avg_ 10
$hypexp set cov_ 2
# Draw values
$hypexp value
```

24.11.2004

20

Creating the network

- Nodes
 - set n0 [\$ns node]
 - set n1 [\$ns node]
- Links and queuing
 - \$ns duplex-link \$n0 \$n1 <bandwidth> <delay> <queue_type>
 - <queue_type>: DropTail, RED, CBQ, FQ, SFQ, DRR
 - example: link with 10 Mbps, 10 ms delay, buffer size 100, RED buffer control

```
$ns duplex-link $n0 $n1 10Mbps 10ms RED
# Set queue size
$ns queue-limit $n0 $n1 100
# Set RED parameters
set redq [$ns link $n0 $n1 queue]
$redq set thresh_ 0
$redq set maxthresh_ 100
$redq set linterm_ 20
$redq set mean_pktsize_ 500
$redq set q_weight_ 0.001
```

24.11.2004

21

Creating connections: UDP

- UDP
 - set udp [new Agent/UDP]
 - set null [new Agent/Null]
 - \$ns attach-agent \$n0 \$udp
 - \$ns attach-agent \$n1 \$null
 - \$ns connect \$udp \$null
- All above combined into one command:
 - \$ns create-connection <src_type> <src_node> <dst_type> <dst_node> <packet_class>
 - \$ns create-connection UDP \$n0 Null \$n1 1

24.11.2004

22

Creating traffic: on top of UDP

- CBR
 - Constant Bit Rate
 - set src [new Application/Traffic/CBR]
- Exponential or Pareto on-off
 - on/off times exponentially/Pareto distributed
 - set src [new Application/Traffic/Exponential]
 - set src [new Application/Traffic/Pareto]
- Connecting application to transport
 - “\$udp” defined earlier
 - \$src attach-agent \$udp
- Above are only traffic sources for a single user
 - ns2 does not provide much support for generating background (aggregate) traffic
 - for example, generating pure GI/GI/1 – type traffic needs to be done “manually” (either at C++ or OTcl level)

24.11.2004

23

Creating Connection: TCP

- TCP
 - set tcp [new Agent/TCP]
 - set tcpsink [new Agent/TCPSink]
 - \$ns attach-agent \$n0 \$tcp
 - \$ns attach-agent \$n1 \$tcpsink
 - \$ns connect \$tcp \$tcpsink
- ... or above in one command:
 - \$ns create-connection TCP \$n0 TCPSink \$n1 1
- Different TCP variants:
 - TCP = Tahoe TCP (slow start, AIMD)
 - TCP/Reno = Reno TCP (above + fast retransmit/fast recovery)
 - TCP/NewReno = modified Reno TCP with improved fast retransmit
 - TCP/Sack1 = SACK TCP (selective ACK)
 - other sources: TCP for asymmetric links (wireless), RTP source, RTCP source
 - different sinks: for each TCP type, LossMonitor (sink with packet loss monitoring)

24.11.2004

24

Creating traffic: on top of TCP

- FTP
 - set ftp [new Application/FTP]
 - \$ftp attach-agent \$tcp
- Telnet
 - set telnet [new Application/Telnet]
 - \$telnet attach-agent \$tcp

24.11.2004

25

Starting/stopping traffic agents

- Starting and stopping times scheduled as events to the scheduler
 - \$ns at <time> <event>
- Starting
 - \$ns at 1.0 "\$ftp start"
 - greedy source (sends infinitely long)
 - similarly for CBR, telnet and on/off sources
- Stopping
 - \$ns at 5.0 "\$ftp stop"
 - similarly for CBR, telnet and on/off sources
- Sending for example 1000 packets
 - \$ns at 7.0 "\$ftp produce 1000"
 - works only for FTP

24.11.2004

26

Creating Traffic: Trace Driven

- Trace driven
 - set tfile [new Tracefile]
 - \$tfile filename <file>
 - set src [new Application/Traffic/Trace]
 - \$src attach-tracefile \$tfile
- <file>:
 - each record consists of two 32 bit fields
 - inter-packet time (msec) and packet size (byte)

24.11.2004

27

Tracing

- Trace packets on all links of the network
 - \$ns trace-all [open test.out w]
- Turn on tracing on specific links
 - \$ns trace-queue \$n0 \$n1
- Trace format:


```
+ 0.89456 0 2 cbr 210 ----- 0 0.0 3.1 0 0
- 0.89456 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
```

 - event type: (enqueue = +, deque = -, receive = r, drop = d)
 - event time
 - node ids of traced link (2 fields)
 - name of packet ("source's name")
 - packet size
 - flags (not used here)
 - flow identifier
 - source/destination addresses (2 fields)
 - sequence number
 - unique packet identifier (all packets created in the simulation have a unique id)

24.11.2004

28

Monitoring

- Sometimes tracing produces “too much” data
 - e.g., just want to know number of arrivals or dropped packets on a link or per flow
 - ⇒ monitors
- Queue monitors
 - set qmon [\$ns monitor-queue \$n0 \$n1]
 - to read number of packet arrivals and drops
 - set parr [\$qmon set parrivals_]
 - set drops [\$qmon set pdrops_]
- Flow monitors
 - enable flow monitoring
 - set fmon [\$ns makeflowmon Fid]
 - \$ns attach-fmon [\$ns link \$n0 \$n1] \$fmon
 - count arrivals and drops for flow with id xx
 - set fclassifier [\$fmon classifier]
 - set flow1 [\$fclassifier lookup auto 0 0 xx]
 - set parr [\$flow1 set parrivals_]
 - set pdrops [\$flow1 set pdrops_]

24.11.2004

29

Summary: generic script structure

```

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

```

24.11.2004

30

Where to look for information?

- NS2 manual
 - <http://www.isi.edu/nsnam/ns/ns-documentation.html>
 - big document, can download into own directory to make accessing faster
- Daily snapshot of the class hierarchy
 - <http://www-sop.inria.fr/planete/software/ns-doc/ns-current/>
 - good source of information, can see the whole class hierarchy with one “snap shot”

24.11.2004

31

If you need to view the C++/OTcl code...

- Viewing code is one way to find out how things work
 - manuals often don't explain everything
 - want to see, e.g., what variables are visible in OTcl from C++
- All paths given here are relative to your ns2 top directory
 - here we assume it is ns-allinone-2.1b9a
- C++ code:
 - /ns-allinone-2.1b9a/ns-2.1b9a/
- OTcl
 - /ns-allinone-2.1b9a/ns-2.1b9a/tcl/lib
 - ns-default.tcl (contains all default values of ns2-objects)
 - also OTcl definitions of many other basic objects used during simulations
 - /ns-allinone-2.1b9a/ns-2.1b9a/tcl
 - most specialized objects under sub-directories

24.11.2004

32

Other functionality, but not covered here...

- In ns2
 - link level errors
 - LAN simulations (including wireless)
 - routing
 - multicast
 - Mobile IP
 - DiffServ
- Visualization tools
 - nam-1 (Network AniMator Version 1)
 - packet-level animation
 - well supported by ns
 - xgraph
 - conversion from ns trace to xgraph format