Network simulators: ns2

Lecturer: Dmitri A. Moltchanov
E-mail: moltchan@cs.tut.fi

http://www.cs.tut.fi/kurssit/TLT-2707/
• Overview;
• Architecture;
• Basics of simulations;
• nam;
• xgraph;
• Example.
1. Overview

Characteristics:
- discrete-event;
- network simulator;
- basically TCP/IP networks;
- wired and wireless components included.

Intended usage:
- research;
- development;
- education.

Developing:
- research institutes and universities;
- freely distributed and open source.
1.1. History of developing

Briefly:

- 1989: REAL simulator by UCB;
- 1990: ns1: LBL;
- 1995: ns2 DARPA VINT project (Virtual InterNet Testbed)
- currently: ns2 supported by DARPA projects.

Current status:

- ns2 runs on:
  - almost all UNIX and Linux and Win 95/98/2000/XP.
- current stable version: 2.31 released in Sep. 7, 2007
  - roughly 1 release in 6 months + daily snapshots.
- http://www.isi.edu/nsnam/ns/ or http://nsnam.isi.edu/nsnam/index.php/Main_Page
- > 200000 line of codes; > 1000 institutions; > 10000 users; > 400 pages of short manual;
1.2. Components

**ns2 distribution consists of:**

- *ns2 itself;*

- nam: Network AniMator
  - visualizing ns2 output;
  - GUI for simple scenarios.

- Mandatory support tools:
  - tcl/tk: script language
  - otcl: object-oriented tcl;
  - TclCL: tcl library.

- Pre-processing tools:
  - traffic, topology generators, converters.

- Post-processing tools:
  - trace analysis: awk, sed, perl, tcl.
  - very simple and not recommended.
1.3. Installation

Differs for:

- Unix/Linux;
- Windows.

**Two types are available:**

- Via compilation;
  - get ’all-in-one package’;
  - get pieces and then compile.
- Obtaining binaries.
  - easiest way to run on Win platform.

**Hints and notes:**

- to compile on Windows you need Cygwin (Unix emulation);
- Size of ’all-in-one’ is around 320Mb (v2.29).
Building from pieces (Unix/Linux):

- get Otcl, TclCL and ns2;
- unpack in some temp folder at the same level;
- cd into the OTcl directory;
  - run ./configure;
  - run make;
- cd into the TclCL directory;
  - run ./configure;
  - run make;
- cd into the ns directory;
  - run ./configure;
  - run make;
- Verify that it built correctly and runs by running ./validate.

Note: ns2 'all-in-one' contains 'install' script (just run it).
1.4. Support

Documentation:

- mailing list: ns-users@isi.edu
  - to get in send message to the address with ”subscribe ns-users” in the body;
  - to browse archive go to http://www.isi.edu/nsnam/ns/.

- ns manual:
  - available at http://www.isi.edu/nsnam/ns/;
  - is only a short reference guide.

- tutorials:
  - a number is available on http://www.isi.edu/nsnam/ns/;
  - faq: http://www.isi.edu/nsnam/ns/ns-faq.html;
  - de-facto tutorial: http://www.isi.edu/nsnam/ns/tutorial/index.html;
  - installation and bug fixes: http://www.isi.edu/nsnam/ns/ns-problems.html;
2. Ns 2 architecture

**Object-oriented structure:**

- advantage: code reuse;
- shortcomings: performance.

**Software structure:**

- uses two languages to separate control and processing:
  - C++: packet processing;
  - Otcl: control.

- Packet processing:
  - C++ makes it fast and detailed;
  - C++ makes it scalable.

- Control:
  - Otcl makes it easy to create scenarios;
  - Otcl makes it easy to understand third party scripts.
2.1. Scalability and extensibility

**Scalability:**
- per-packet processing must be fast;
- separating control and packet handling.

**Extensibility:**
- must be easy to add new objects;
- object trees to understand hierarchy:
  - in C++;
  - in Otcl.
- C++ and Otcl trees are split:
  - if not needed nothing have to be changed at a certain level;
  - helps a lot!
- Otcl class hierarchy: http://www-sop.inria.fr/planete/software/ns-doc/ns-current/;
2.2. Simple script

.tcl
create the simulator object and assign it name “ns”
ns [new Simulator]

schedule event at time 1 to print Hello World!
at 1 “puts \“Hello World!\””

.. and exit at time 1.5
at 1.5 “exit”

un the simulation
run

v 74% ns simple.tcl
world!
v 75%
2.3. Tcl basics

**Tcl language:**

- semantics is similar to perl;
- one can check tutorial at: http://www.msen.com/%7Eclif/TclTutor.html;
- real programming language to create network topology;
- tcl is used by Otcl to construct advanced objects.

**Tcl contains:**

- lists, arrays, associative arrays etc.;
- procedures and functions;
- flow controls: if, while, for, etc.
**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
}
2.4. Otcl basics

```otcl
on
:or
:proc init {age} {
stvar age_
$age
}

:proc greet {} {
stvar age_
ge_years old: How 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]
那个人 greet
$kid greet
```

- codes can be reused (for example, different version of TCP).
2.5. Viewing source code

Where to find:

- **C++:**
  - `/whereYouInstalled/ns-(ver)/`
  - where (ver) is your version of ns2 (e.g. 2.1b9a).

- **Otcl:**
  - `/whereYouInstalled/ns-(ver)/tcl/lib/`
    - *ns-default.tcl*: default values for ns2 objects;
    - *other Otcl definitions*.
  - `/whereYouInstalled/ns-(ver)/tcl/`
    - *specialized objects in subdirs.*
3. Simulations

Basic steps:

• Create the simulation environment;
  – event list, scheduler, etc.;

• Create the network:
  – nodes and links between them.

• Create connections:
  – TCP, UDP (in some sense).

• Create applications:
  – CBR flow, FTP, WWW traffic.

• Trace network elements:
  – trace queue, trace flow.
3.1. Creating environment

The following are the common commands:

- Create event scheduler:
  
  - `set ns [new Simulator];`

- Schedule events:
  
  - `$ns at <time> <event>;`

- Start scheduler:
  
  - `$ns run.`

- Stop and close everything:
  
  - `$ns at $x "exit".`
  
  - Where `$x` is some instant of time.

Note: before closing you must take additional actions:

- flush all traces to files;

- close all files.
3.2. Generating RVs

Create new generator:

- set rng [new RNG];
- $rng$ seed 0.

RNs from other distributions:

- using class RNG:
  - uniform: $rng$ uniform a b.
- using class RandomVariable:
  - distributions: uniform, exponential, hyperexponential, Pareto;

```bash
set hypexp [new RandomVariable/HyperExponential]
hypexp set avg_ 10
hypexp set cov_ 2
Draw values
hypexp value
```
3.3. Creating the network

Nodes:

- set n0 [$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

```bash
$ns duplex-link $n0 $n1 10Mbps 10ms RED
# Set queue size
$nssim 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
```
3.4. Creating connections

Creating UDP flow

```
set udp [new Agent/UDP]
set null [new Agent/Null]
$ns attach-agent $n0 $udp
$ns attach-agent $n1 $null
$ns connect $udp $null
```

In one command:

- `$ns create-connection <src_type> <src_node> <dst_type> <dst_node> <packet_class>;`
- example: `$ns create-connection UDP $n0 Null $n1 1`.

Other sources:

- TCP (we will consider);
- RTP source, RTCP source.
- TCP for wireless links.
Creating TCP flow

```
set tcp [new Agent/TCP]
set tcpsink [new Agent/TCPSink]
$ns attach-agent $n0 $tcp
$ns attach-agent $n1 $tcpsink
$ns connect $tcp $tcpsink
```

**In one command:**

- `$ns create-connection <src_type> <src_node> <dst_type> <dst_node> <packet_class>;`

- example: `$ns create-connection TCP $n0 TCPSink $n1 1`.

**Some included TCP versions:**

- TCP: Tahoe TCP (slow start and AIMD);
- TCP/Reno: Reno TCP (... + fast retransmit/fast recovery);
- TCP/NewReno: TCP Reno with improved fast retransmit;
- TCP/Sack: TCP SACK (Selective ACK).
3.5. Creating traffic on top of UDP

**CBR:**
- Constant bit rate;
- set src [new Application/Traffic/CBR].

**Exponential or Pareto ON/OFF:**
- on/off times are exponentially distributed;
- set src [new Application/Traffic/Exponential];
- set src [new Application/Traffic/Pareto].

**Connecting to transport:**
- $udp$ defined earlier;
- $src$ attach-agent $udp$. 
ns2 includes support for traffic traces:

```plaintext
set tfile [new Tracefile]
$tfile filename <file>
set src [new Application/Traffic/Trace]
$ssrc attach-tracefile $tfile
```

**What <file> should look like:**

- each record consist of two 32 bit field;
- inter-packet time (msec) and packet size (in bytes).

**For example:** there is converter for MPEG frame size file to ns2.
3.6. Creating traffic on top of TCP

**FTP:**

- set ftp [new Application/FTP];

- attaching to TCP: $ftp attach-agent $tcp.

**Telnet:**

- set telnet [new Application/Telnet];

- attaching to TCP: $telnet attach-agent $tcp.
3.7. Starting/stopping traffic sources

Starting and stopping times scheduled as events to the scheduler:

- $\text{ns at } \langle \text{time} \rangle \langle \text{event} \rangle$.

Starting:

- $\text{ns at 1.0 } "\text{ftp start}"$;
- sends infinitely long;
- the same for CBR, telnet and on/off sources.

Stopping:

- $\text{ns at 5.0 } "\text{ftp stop}"$;
- the same for CBR, telnet and on/off sources.

Sending for example 1000 packets:

- $\text{ns at 7.0 } "\text{ftp produce 1000}"$;
- works for FTP only.
3.8. Tracing

Trace packets on all links of the network:

- `$ns trace-all [open test.out w]`.

Tracing on specific links:

- `$ns trace-queue $n0 $n1`.

```
+ 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
```

Type: (enqueue = +, dequeue = -, receive = r, drop = d)

IME

Is of traced link (2 fields)

If packet ("source’s name")

Size

Not used here

Antifer

Destination addresses (2 fields)

Ice number

Packet identifier (all packets created in the simulation have a unique id)
3.9. Monitoring

**Queue monitor:**

- set qmon [\$ns monitor-queue \$n0 \$n1];

- for packet arrivals and drops:
  - set parr [\$qmon set parrivals.];
  - set drops [\$qmon set pdrops.].

**Flow monitor:**

- enable flow monitoring:
  - set fmon [\$nssim makeflowmon Fid];
  - \$nssim attach-fmon [\$nssim link \$n0 \$n1] \$fmon.

- count arrivals and drops for flow with id xx:
  - set fclassifier [\$fmon classifier];
  - \$fclassifier lookup auto 0 0 xx;
  - set parr [\$flow1 set parrivals.]; set pdrops [\$flow1 set pdrops.].
3.10. Generic methodology

::et 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
3.11. Other functionalities

ns2 also provides:

- errors on the data-link layer;
- LAN and WLAN;
- Routing;
- Multicasting;
- Mobile IP;
- DiffServ;
- MPLS;
- ...
4. nam

Nam (Network AniMator):

- packet-level animation;
- almost integrated with ns2.
Lecture: Ns2 simulator
5. Xgraph

Allows to make graphs:

- frequently used with ns2;
- simple visualizer.