IASTED ASM 2004 Tutorial:

The *ns-2* Network Simulator

Michael Welzl  [http://www.welzl.at](http://www.welzl.at)

Distributed and Parallel Systems Group
Institute of Computer Science
University of Innsbruck, Austria
Outline

- ns-2 overview
- Writing simulation scripts
- Obtaining results
- General network simulation hints
- Teaching with ns-2
- Conclusion

simulated transit-stub network (ns/nam dump)
ns-2 overview
Some thoughts about network simulation

- Real-world experiments sometimes impossible
- Mathematical modeling can be painful
  - Sometimes even painful and inaccurate!
  - Stack processing delay, buffering, timing, ..
- Simulation cannot totally replace real-world experiments:
  - Scaling, simulated vs. real time (long simulations), ..
- and vice versa!

- Simulation sometimes misses the whole picture
  - Focus on specific layers, wrong assumptions, ..
ns-2 (ns version 2) overview

- discrete event simulator (but does not strictly follow DEVS formalism)

- OTcl and C++
  - OTcl scripts, C++ mechanism implementations, timing critical code

- de facto standard for Internet simulation

- open source! advantages:
  - facilitates building upon other work
  - allows others to use your work

- disadvantages:
  - huge and complicated code
  - partially missing, somewhat poor documentation
What to use it for

• Mainly research
  - Examining operational networks, e.g. with Cisco brand names: OPNET

• Focus on Internet technology
  - the tool for TCP simulations

• Not ideal for new link layer technologies etc.
  - typically neglects everything underneath layer 3
  - there are special simulators for things like ATM

• But: lots of features!
  - CSMA/CD, 802.11 now available ... if you really want it
  - Some things must be added manually ("Contributed Modules")
Some ns-2 features

- Lots of TCP (and related) implementations
- Mobility
- Satellite nodes: specify longitude, latitude, altitude
- Internet routing protocols
- Various link loss models
- Various traffic generators (e.g., web, telnet, ..)
- Network emulation capability (real traffic traverses simulated net)

flexible, but hard to use; alternatives: NISTNet, Dummynet
Simulation process

- Not interactive

- Script describes scenario, event scheduling times

- Typical last line: `$ns run`

- ns generates output files

- "Interactive simulation": view `.nam` output file with network animator `nam`

- `.nam` files can become huge!
Typical long-term simulation process

- Mechanism implementation in C++, test with simple OTcl script
- Simulation scenario development in OTcl
- Test simulation with nam

One way to obtain a useful result:
perl script -> simulations with varying parameter(s) -> several output files -> perl script -> result file -> xgraph (gnuplot, MS Excel, ..) -> ps file
Simulation elements

- **Code:** C++ network elements, OTcl simulator elements and simulation script, perl (or whatever) scripts for running simulation and interpreting results
- **ns** to run simulation, **nam** for immediate feedback
- **Visualization tool** (xgraph, gnuplot, ..)

![Diagram of simulation elements]

- **C++ network elements**
  - **Otcl script**
  - **ns trace file**
  - **visualisation tool**
  - **diagram**

- **nam**
  - **nam trace file**
  - **user**
  - **perl script**
  - **results**

→ generates

→ uses
Writing simulation scripts
Writing simulation scripts

- **OTcl** - what you require is some Tcl knowledge
  - not complicated, but troublesome:
    - e.g., line breaks can lead to syntax errors

- Some peculiarities
  - variables not declared, generic type: \texttt{set a 3} instead of \texttt{a=3}
  - expressions must be explicit: \texttt{set a [expr 3+4]}
    - \texttt{set a 3+4} stores string "3+4" in a
  - use "$" to read from a variable: \texttt{set a $b} instead of \texttt{set a b}
    - \texttt{set a b} stores string "b" in a

- Write output: \texttt{puts "hello world"}

- OTcl objects: basically used like variables
  - method calls: \texttt{$myobject method param1 param2}
# Create a simulator object
set ns [new Simulator]

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

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

set variable value
[new Simulator] generates a new Simulator object
open "out.nam" for writing, associate with nf
$ns ns-command parameters
log everything as nam output in nf
Run nam with parameter out.nam as background process (unix)

More about Tcl & OTcl:
http://www.isi.edu/nsnam/NS/tutorial/index.html
("finding documentation")
# Insert your own code for topology creation
# and agent definitions, etc. here

## Code template /2

```nsc
# Create two nodes
set n0 [$ns node]
set n1 [$ns node]

# Create a duplex link between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail

# Create a CBR agent and attach it to node n0
set cbr0 [new Agent/CBR]
$ns attach-agent $n0 $cbr0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005

# Create a Null agent (a traffic sink) and attach it to node n1
set sink0 [new Agent/Null]
$ns attach-agent $n1 $null0

# Connect the traffic source with the traffic sink
$ns connect $cbr0 $sink0

# Schedule events for the CBR agent
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
```

Other agents: TCP, UDP, ...

Agent parameters

CBR <-> Null, TCP <-> TCPSink!

"$cbr0 start" contains no destination address

# Call the finish procedure after 5 seconds simulation
$ns at 5.0 "finish"

# Run the simulation
$ns run

$ns at time "command"
schedule command
Applications

- Apps: layered on top of agents

- CBR the "clean" way:
  
  ```
  set udp [new Agent/UDP]
  $ns attach-agent $myNode $udp

  set cbr [new Application/Traffic/CBR]
  $cbr set packetSize_ 500
  $cbr set interval_ 0.005
  $cbr attach-agent $udp

  $ns connect $udp $myNullAgent

  $ns at 0.5 $cbr start
  $ns at 4.5 $cbr stop
  ```

Other applications:
- ftp (greedy tcp source)
- telnet
- Traffic/Exponential
- Traffic/Pareto

or LossMonitor

always connect agents, sometimes also apps!

start / stop the app, not the agent!
More agents

• CBR <-> Null, CBR <-> LossMonitor
  - note: CBR agent deprecated!

• TCP <-> TCPSink
  - one-way Tahoe implementation
  - other flavours: TCP/Reno, TCP/NewReno, TCP/Sack1, TCP/Vegas
  - two-way: use FullTCP at both sides

• TCP-friendly connectionless protocols:
  - RAP <-> RAP
    Rate Adaptation Protocol (rate-based AIMD + Slow Start)
  - TFRC <-> TFRCSink
    TCP Friendly Rate Control protocol (based on TCP throughput-equation)

requires TCPSink/Sack1!

Use app on top

$app start
$app stop

$agent start
$agent stop
More about links

- **Link creation:**
  - `$ns_dupelex-link node1 node2 bw delay qtype args`
  - *qtypes:* DropTail, FQ, SFQ, DRR, CBQ, RED, RED/RIO, ...

- **Link commands:**
  - `set mylink [$ns link $node1 $node2]`
  - `$mylink command`

- **Queue parameters:**
  - `set myqueue [$mylink queue]`
  - `$myqueue set parameter_ value` or `$myqueue command`

**Note:** for all default parameters, see:
ns/tcl/lib/ns-default.tcl
Transmitting user data

- **TcpApp** application - use on top of **Agent/TCP/SimpleTCP**

  - `set tcp1 [new Agent/TCP/SimpleTCP]
    set tcp2 [new Agent/TCP/SimpleTCP]
    $ns attach-agent $node1 $tcp1
    $ns attach-agent $node2 $tcp2
    $ns connect $tcp1 $tcp2
    $tcp2 listen
    ( $tcp1 listen )`  
      
    - `set app1 [new Application/TcpApp $tcp1]
      set app2 [new Application/TcpApp $tcp2]
      $app1 connect $app2
      ( $app2 connect $app1 )`  
      
    - `$ns at 1.0 "$app1 send 100 "$app2 app-rev 123""`  
      
    - `Application/TcpApp instproc app-rev { number } { ... }`
Writing your own protocol

• simple (for teaching): use TcpApp
  - inefficient, not very flexible

• more useful (for researchers): change ns code

• two basic approaches
  1. truly understand architecture
     (class hierarchy, including “shadow objects” in OTcl etc.)
  2. “hack” code
     - change existing code to suit your needs
     - generally use C++ (and not OTcl) if possible

works surprisingly well; I recommend this.
How to "hack" ns code

- "ping" example in the Marc Greis tutorial (easy - try it!)
  - simple end-to-end protocol

how to integrate your code in ns:
  - write your code: e.g., ping.h and ping.cc
  - if it's an agent...
    - "command" method = method call from OTcl
    - recv method = called when a packet arrives
  - if it's a new packet type: change packet.h and tcl/lib/ns-packet.tcl
  - change tcl/lib/ns-default.tcl
    - define and initialize attributes that are visible from OTcl
  - do a "make depend", then "make"

Note: ns manual mixes "how to use it" with "what happens inside"
  - this is where you look for details
Obtaining results
Obtaining results

Use LossMonitor instead of Null agent

```tcl
proc record {} {
    global sink0 ns f0
    # Set the time after which the procedure should be called again
    set time 0.5
    # How many bytes have been received by the traffic sinks?
    set bw0 [sink0 set bytes_]
    # Get the current time
    set now [ns now]
    # Calculate the bandwidth (in MBit/s) and write it to the files
    puts $f0 "$now [expr $bw0/$time*8/1000000]"
    # Reset the bytes_ values on the traffic sinks
    sink0 set bytes_ 0
    # Re-schedule the procedure
    ns at [expr $now+$time] "record"
}
```

**Important:**

$ns at 0.0 "record"

**Note:**

puts "[expr 1+1]" -> 2
puts "1+1" -> 1+1
Obtaining results /2

- LossMonitor: simple solution for CBR, but TCP <-> TCPSink!

- Very common solution: generate tracefile
  
  ```
  set f [open out.tr w]
  $ns trace-all $f
  ```

- Trace file format:
  
  event | time | from | to | type | size | flags | flow ID | src addr | dst addr | seqno | uid

  event: "r" receive, "+" enqueue, "-" dequeue, "d" drop
  type: "cbr", "tcp", "ack", ...
  flags: ECN, priority, ...
  flow id: similar to IPv6
  uid: unique packet identifier

  `Color your flow in nam: $agent set class_ num (e.g. $udp set class_ 1)`
  `Color your flow in nam: $ns color num color (e.g. $ns color 1 blue)`
  `Monitoring queues in nam: $ns duplex-link-op $node1 $node2 queuePos 0.5`
Obtaining results /3

- Problem: trace files can be very large

- Solution 1: use pipe
  
  ```
  set f [open | perl filter.pl parameters w]
  $ns trace-all $f
  ```

- Other solutions:
  - Use specific trace or monitor - more efficient!
  - direct output in C++ code - even more efficient, but less flexible!

- Note: TCP throughput != TCP "goodput"
  - Goodput: bandwidth seen by the receiving application
  - should not include retransmits!
  - requires modification of TCP C++ code
Dynamic vs. long-term behavior

- Typically two types of simulation results:
  - dynamic behavior
  - long-term behavior: 1 result per simulation (behavior as function of parameter)

- Dynamic behavior: parse trace file (e.g. with "throughput.pl")
  - roughly:
    ```
    oldtime = trace_time;
    while (read != eof)
      if (trace_time - oldtime < 1)
        sum_bytes += trace_bytes;
      else {
        print "result: sum_byte / (trace_time - oldtime);"
        oldtime = trace_time;
      }
    ```
  - actually slightly more difficult - e.g., "empty" intervals should be included

- long-term behavior
  - script calls simulations with varying parameter
  - each simulation trace file parsed (simply summarize data or calculate statistics)
Long-term behavior

- **Simulation script:**
  
  ```
  (..)
  set parameter [lindex $argv 0]
  (..)
  set f [open "| perl stats.pl tcp w"]
  $ns trace-all $f
  (..)
  ($parameter used in simulation!)
  ```

- **loop:**
  
  ```
  for {set i 1} {$i <= 10} {set i [expr $i+1]} {
      exec echo $i >> results.dat
      exec ns sim.tcl $i >> results.dat
      exec ns sim.tcl $i >> results.dat
  }
  ```

- **Plot final file with additional tool (e.g. gnuplot)**
Visualizing dynamic behavior: CADPC Routing robustness

- Manual edit

- xgraph plot of throughput.pl result

- nam output ("print" + postscript driver)

- Manual edit
Example result 2: CADPC vs. TCP, dynamic
Example 3: CADPC startup enhancement
Long-term CADPC results (gnuplot)
General network simulation hints
Recommendations

- Select parameters carefully
  - consider RTT: what is realistic? what is a reasonable simulation duration?
  - ideally, duration should depend on statistics
    • e.g., until variance < threshold
    • not supported by ns
    • thus, just simulate long (realistic duration for, e.g., ftp download)

- Frequently ignored (but important) parameters:
  - TCP maximum send window size (should not limit TCP behavior)
  - TCP “flavor” (should be a state-of-the-art variant like SACK)
  - buffer length (should be bandwidth*delay)
  - Active Queue Management parameters (RED = hard to tune!)
Recommendations /2

- Start simple, then gradually approach reality
  - 1 source, 1 destination
  - multiple homogeneous flows across single bottleneck
  - multiple homogeneous flows with heterogeneous RTTs
  - multiple heterogeneous flows with homogeneous and heterogeneous RTTs
  - impact of routing changes, various types of traffic sources (web, “greedy”, ..)

- eventually implement and test in controlled environment
- …and then perhaps even test in real life :)

<table>
<thead>
<tr>
<th>Abstraction level</th>
<th>No. users</th>
<th>RTTs</th>
<th>No. resources</th>
<th>Traffic model</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>equal</td>
<td>1</td>
<td>fluid</td>
<td>maths</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>equal</td>
<td>1</td>
<td>fluid</td>
<td>vector diagrams</td>
</tr>
<tr>
<td>3</td>
<td>n</td>
<td>equal</td>
<td>1</td>
<td>fluid</td>
<td>maths</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>heterogeneous</td>
<td>1</td>
<td>fluid</td>
<td>CAVT</td>
</tr>
<tr>
<td>5</td>
<td>n</td>
<td>heterogeneous</td>
<td>1</td>
<td>discrete</td>
<td>“normal” simulation</td>
</tr>
<tr>
<td>6</td>
<td>n</td>
<td>heterogeneous</td>
<td>m</td>
<td>discrete</td>
<td>“normal” simulation</td>
</tr>
<tr>
<td>7</td>
<td>n</td>
<td>heterogeneous</td>
<td>m</td>
<td>discrete</td>
<td>real life experiments</td>
</tr>
</tbody>
</table>
Common topologies

- **Dumbbell:**
  evaluate basic end2end behaviour, TCP-friendliness, ..
  - often: n vs. m flows

- **Parking Lot:**
  evaluate behaviour with different RTT's, fairness, ..
  - often: S0/D0 - n flows, all other S/D pairs - m flows
Recommended papers (simulation method)


- and this website: http://www.icir.org/models/bettermodels.html
Teaching with ns-2
Teaching experience: ns for lab exercises

- **Reason to use ns**
  - lack of other tools to “fool around” with network dynamics (mainly congestion control)
  - some sort of “hands-on experience” without endangering real net

- **Personal experience**: (3 times; lab courses accompanying networks lecture) - Students did not like it!
  - poor documentation (problem eliminated for 3rd try, didn’t help)
  - OTcl learning effort large, but exercises small - too much overhead
  - some things do not work as expected; bugs etc.

- **Recommendations**
  - avoid long-term simulation (various languages and files for single exercise)
  - avoid C++ changes; exercises in OTcl only, with precompiled ns-2

- **Currently working on a GUI!** (monitor my ns website)
Example exercise 1: congestion collapse

- Congestion control necessary

- adding fast links does **not** help!

![Diagram](image_url)

- total throughput w/o cc.: 20kb/s
- total throughput w/ cc.: 110kb/s
Example exercise 1: congestion collapse /2

Goal: operation at the "knee"
Internet congestion control: History

- **1968/69**: dawn of the Internet
- **1986**: first congestion collapse
- **1988**: "Congestion Avoidance and Control" (Jacobson) Combined congestion/flow control for TCP

- **Goal**: stability - in equilibrium, no packet is sent into the network until an old packet leaves
  - ack clocking, "conservation of packets” principle
  - made possible through window based stop+go - behaviour

- Superposition of stable systems = stable →
  network based on TCP with congestion control = stable
TCP Congestion Control /1: Tahoe, 1988

- Distinguish:
  - **flow control**: protect receiver against overload
    (receiver "grants" a certain amount of data ("receiver window") )
  - **congestion control**: protect network against overload
    ("congestion window" (cwnd) limits the rate: min(cwnd,rwnd) used! )

- Flow/Congestion Control combined in TCP. Several algorithms:

  - (window unit: SMSS = Sender Maximum Segment Size, usually adjusted to Path MTU; init cwnd\leq 2 (*SMSS), ssthresh = usually 64k)
    - **Slow Start**: for each ack received, increase cwnd by 1
      (exponential growth) until cwnd \geq ssthresh
    - **Congestion Avoidance**: each RTT, increase cwnd by SMSS*SMSS/cwnd
      (linear growth - "additive increase")
TCP Congestion Control /2

- If a packet or ack is lost (timeout, roughly 4*rtt), set cwnd = 1, ssthresh = current bandwidth / 2 ("multiplicative decrease") - exponential backoff

- Several timers, based on RTT; good estimation is crucial!

- Later additions: (TCP Reno, 1990) Fast retransmit / fast recovery (notify sender of loss via duplicate acks)
Example exercise 2: TCP vs. UDP
Example exercise 2: TCP vs. UDP /2

- Results by a student in Linz, Austria
Conclusion
Conclusion

- Research
  + freely available
  + de facto standard for (Internet-) simulations
  + open source spirit: compare your mechanism with the state-of-the-art
  + lots of features
    - rather complex architecture; new protocol = typically a hack
    - must use lots of tools (and typically languages) at once

- Teaching
  - OTcl learning effort for exercises
  + nam animations may be good for lectures (repository available)
Key links

- **Official website:** [http://www.isi.edu/nsnam/ns](http://www.isi.edu/nsnam/ns)
  - docs: "ns manual", "Marc Greis’ tutorial", "ns by example"
    - note: HTML version of "ns manual" not always up-to-date
  - large number of "Contributed Modules"

- **Personal ns website:** [http://www.welzl.at/tools/ns](http://www.welzl.at/tools/ns)
  - "throughput" and "stats" scripts
  - working versions of ns and nam Windows binaries
  - German ns documentation (student work)
  - these slides

- **Lloyd Wood’s site:** [http://www.ee.surrey.ac.uk/Personal/L.Wood/ns/](http://www.ee.surrey.ac.uk/Personal/L.Wood/ns/)
  - additional links
Good luck!