2nd ISSNSM’s Tutorial on

Simulating Networks with Network Simulator 2 (ns-2)

(Tutorial T2)

Speaker:
Frank Eyermann

June 3, 2008
ISSNM program chaired by Burkhard Stiller, David Hausheer, University of Zürich

ISSNM laboratory organization chaired by Cristian Morariu, Peter Racz, University of Zürich
ToC

- Structure
- Main parts of a simulation
- Creating a simulation script
  - Exercises 1
- Tracing and Monitoring
  - Exercises 2
- Analyzing traces
  - Exercises 3
ToC

- **LANs**
  - Exercises 4
- **Unicast Routing and Network Dynamics**
  - Exercises 5

- **Outlook**
  - What had to be skipped in this tutorial

---

**Introduction**

- **Frank Eyermann**
  - Frank.Eyermann@unibw.de
  - Information Systems Laboratory
  - Faculty for Computer Science
  - Universität der Bundeswehr, Munich
**Basics (1/3)**

- **ns-2**
  - Discrete event driven simulator
  - (O)Tcl-script describes simulation flow
  - For all kinds of packet-based networks
  - Most Unix-like systems / cygwin
  - Packets are only events
    - No real data (payload) is transferred!
  - Each packet is simulated
    - Scalability issue!
    - “Careful” logging

**Basics (2/3)**

- **Support for**
  - TCP
  - Routing
  - Multicast-Protocols
  - Wired
  - wireless (WLAN and satellite) networks
  - Energy and movement models

- **Lots of extensions in the internet**
  - Mostly badly maintained
  - Only for one special ns-2 version
**Basics (3/3)**

*License Simulator*
- Freely distributable, Open-Source

*Modules*
- Type of license different, depending on author
- mainly:
  - GNU GPL (GNU General Public License)
  - Berkley similar license
  - modified Berkley license
  - And compatible (e.g. Apache 2.0)
- Use without any warranty

---

**Simulation Control**

*With OTcl-Script*
- Describes network topology and configuration
  - nodes
  - links
  - protocols
  - applications
- Describes simulation flow (Course of actions)
  - Starting and stopping of data sources
  - Loss of communication
  - Duration of simulation
  - Firing of periodical events
“Data processing“

- **ns-2 does not transport any data!**
  - Simulated by events
  - Bigger packets arrive “later”
- **Works on packet level**
  - Every single packet is simulated
- **Everything what happens for each packet**
  - implemented in C++, Runtime advantages
- **Additionally some helper classes (Algorithms) in C++**

---

**Elements of a simulation**

- **Simulator**
  - Main class
  - Configuration of the simulation
  - Creating objects
  - Creating events and their scheduling
- **Nodes**
  - Nodes in a network (end or intermediate)
  - Attached list of agents (~protocols)
  - Attached list of neighbors (=> Links)
  - Unique ID (~address)
**Elements of a Simulation**

- **Links**
  - connecting nodes („physically“)
  - simplex-, duplex links
  - multiple access LANs, including wireless
  - bandwidth
  - delay
  - queue object
  - different trace-objects
    - enqueue, dequeue,
    - drop
    - receive (implemented in next node)

**Elements of a simulation**

- **Queues**
  - “part of” link
  - store, drop packets if necessary
  - Decide which packet is dropped
    - Drop-tail (FIFO)
    - Random Early Detect (RED)
    - Class Based Queuing (CBQ, priority + Round Robin)
    - Several Fair Queuing mechanisms (SFQ, DRR,...)
  - „Drop Destination“
    - Object all dropped packets are forwarded to
Elements of a simulation

**Agents**
- Endpoint of (logical) connections
- ~ OSI level 3 (network)
- Create and receive packets
- Implement protocols
- Sometimes additional sender and receiver necessary
- TCP, TCPSink in div. "flavors"
- UDP
- RTP, RTCP
- … (see ns manual, Chapter 10.3)

Simplified scenario

```
$ns attach-agent $node $agent
$ns attach-agent $node $agent
$appl attach-agent $agent
$appl attach-agent $agent
$ns connect
$ns connect
```
#Create a simulator object
set ns [new Simulator]

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

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

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

#Run the simulation
$ns run

Place your code here
General approach

- Create the simulator
- Activate tracing
- Create the nodes and topology
- Create the links
- Activate routing
- Chose error model, if necessary
- Create the traffic
- Send application data

Exercise

- Exercise 1.1
- Exercise 1.2
- Exercise 1.3
- Exercise 1.4
- Exercise 1.5
Trace support

- Two types of output
  - namtrace
  - trace

- nam
  - Graphical user interface for “Replaying” the simulation
  - Auto-layout or define node positions in script
  - Color packets
  - Graphical queue monitoring
  - (Limited) editor for creating simulations
  - namtrace only usable for nam

Two ways to collect data:
- Traces with trace objects
- Counters with monitor objects

Trace objects
- For each single packet writes to trace file:
  - Time of arrival, sending, dropping
  - For links or queues

Tracing of all links to one file is easy:
- $ns trace-all $tf
- Not for bigger simulations!
### Example trace file

<table>
<thead>
<tr>
<th>Action</th>
<th>Time</th>
<th>Service</th>
<th>Port</th>
<th>Flags</th>
<th>Sequence</th>
<th>Expected</th>
<th>Delay</th>
<th>Queue Size</th>
<th>Queue Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0.84824</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 29 46</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>0.85408</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 28 44</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>0.85656</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 30 47</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>0.85656</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 30 47</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>0.8624</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 29 46</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>0.86488</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 31 49</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>0.86488</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 31 49</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>0.87072</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 30 47</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>0.8732</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 32 50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>0.8732</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 32 50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>0.87904</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 31 49</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>0.88152</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 33 52</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>0.88152</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 33 52</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>0.88736</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 32 50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>0.88984</td>
<td>2 3 tcp 1040</td>
<td>------</td>
<td>0 0.0</td>
<td>3.0 34 53</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Creating trace objects

**different trace objects**
- Trace/Enque: packet arrival (at a queue)
- Trace/Deque packet departure (at a queue)
- Trace/Drop packet drop (packet delivered to drop-target)
- Trace/Recv packet receive event at the destination node of a link

```bash
$ns create-trace {type file src dst}
```

**Needs to be included in simulation afterwards (Regularly very complex)!**
- Not feasible
Tracing of links

- **Record all events**
  
  \$ns trace-queue <n1> <n2>
  
  \$ns trace-queue $n2 $n3 $tf

- **Record only the drops**
  
  \$ns drop-trace $n2 $n3 [\$ns create-trace Drop $tf $n2 $n3]

Trace support

- **Monitor objects**
  - Count variable (e.g. number of arrived packets or bytes)
  - for all packets or
  - only for one flow (flow monitor)
  - Advantage: Statistical data without long post processing
Monitoring of Queues

- `$ns monitor-queue <n1> <n2>
  <opt:qtrace> <opt:sampleinterval>
  - qtrace: Channel-ID, optional only when Channel-ID was set in trace-all
  - Returns QueueMonitor object
  - Is configured over attached link
  - `$ns simplex-link and $ns duplex-link have no return value
  - Retrieve link object from simulator:
    set monitoredLink [$ns link $n2 $n3]

Monitoring of Queues

- **Time based**
  - All `sampleInterval` seconds
  - `$monitoredLink queue-sample-timeout`
  - `$monitoredLink start-tracing`
  - Uses for output: `sample-queue-size`
  - starts each time new period!

- **Call directly**
  - `$monitoredLink sample-queue-size`
  - returns string:
    - `$meanBytesQ $meanPktsQ $parrivals_ $pdepartures_ $pdrops_ $barrivals_ $bdepartures_ $bdrops_`
Monitoring of Queues

- **Read all values manually (packets)**
  
  $qmon$ set parrivals_ (# packet arrival)
  $qmon$ set pdepartures_ (# p. depart.)
  $qmon$ set pdrops_ (# packet drops)
  $qmon$ set pkts_ (# packet in queue now)

  set pktint [ $qmon$ get-pkts-integrator ]
  $pktint$ set sum_ (Integral of queue size)

- **Read all values manually (Bytes)**
  
  $qmon$ set barrivals_ (bytes arrival)
  $qmon$ set bdepartures_ (bytes depart.)
  $qmon$ set bdrops_ (bytes dropped)
  $qmon$ set size_ (bytes in queue now)

  set byteint [ $qmon$ get-bytes-integrator ]
  $byteint$ set sum_ (Integral queue size)
  $samples$ mean (Average queue delay)
Monitoring of Queues

- Time in queue
  - Samples as extension to QueueMonitor
  - set samples [new Samples]
  - $qMon set-delay-samples $samples

- Samples Klasse
  - Siehe ~ns/tools/integrator.{h | cc}
  - newPoint (delay)
  - cnt
  - mean
  - variance
  - reset

Monitoring of connections

- for UDP connections
  - Special agent
  - Instead of Agent/Null
  - Set Agent/LossMonitor

- Read variables
  - [set $lossMon nlost_] # not arrived packets
  - [set $lossMon npkts_] # arrived packets
  - [set $lossMon bytes_] # arrived bytes
Tracing of RED Queues

- Queue/RED derived from Queue
- additionally
  - ave_ average queue size
  - prob1_ dropping probability
  - curq_ current queue size
  - cur_max_p_ current max. drop prob.
  - set redq [[$ns link $n2 $n3] queue]
    • Queue of link between n2 and n3
  - redq trace variable
    • Variables see above
  - redq attach $tf ($tf=trace file)

Monitoring of Flows

- Special monitor
  - Common and
  - Flow-based information
    set fmon [$ns makeflowmon Fid]
    $fmon attach $monFile
    $ns attach-fmon $link $fmon
    • Fid: Name of a classifier (Classifier/Hash/ )
    • $link: Link object
- Prints values only on call of dump
  - To use Fid classifier have agent mark flows with set class_
Evaluate Traces

- Two applications included
  - nam
  - xgraph

- Other
  - Everything that can handle long tables
  - Delimiter is " " (Blank)
  - Spread sheets unsuitable (Traces too long, e.g., Excel can handle max. 64k lines)
  - Dependencies between lines, requires programming/scripting language
Best practice Trace

- **During simulation**
  - Output only relevant data
  - Less is more!
  - Do not write nam trace if not needed

- **Post-Processing**
  - Filter (e.g., grep)
  - Calculate derived data (e.g., calculate delay from two timestamps)
  - If necessary create new data file
  - Visualize (xgraph or gnuplot)

Best practice Monitor

- **During simulation**
  - Read statistical data periodically from monitor
  - Directly write desired format to file

- **Post-Processing**
  - Filter (e.g., grep)
  - Calculate further derived values
  - Visualize (xgraph or gnuplot)
xgraph

- Part of ns-2
- Data set:
  - Two columns, separated by blanks
  - Empty line starts new data set
  - Or store data sets in separate files
- Lots of possibilities to customize output
  - See `man xgraph`

Xgraph - example

<table>
<thead>
<tr>
<th>TitleText: Sample Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5 7.8</td>
</tr>
<tr>
<td>1.0 6.2</td>
</tr>
<tr>
<td>&quot;set one&quot;</td>
</tr>
<tr>
<td>1.5 8.9</td>
</tr>
<tr>
<td>&quot;set two&quot;</td>
</tr>
<tr>
<td>2.2 12.8</td>
</tr>
<tr>
<td>2.4 -3.3</td>
</tr>
<tr>
<td>2.6 -32.2</td>
</tr>
<tr>
<td>2.8 -10.3</td>
</tr>
</tbody>
</table>

Options as `option: value`

Name of data set, anywhere, starts with "

Empty line or new file starts new data set
gnuplot

- **Plots functions**
  - Available for almost all operating systems
- **Interactive mode**
  - `plot "filename" using RowX:RowY, "filename" using RowX:RowY2 {axes x1y2} {with lines}`
- **Plots even millions of points**
- **man gnuplot**
- **Good tutorials for gnuplot available in Internet**

---

Useful (unix) commands

- **grep**
  - Filters based on simple regular expression
  - `grep -e "regex" filename`
- **sed**
  - Filter with more complex regular expressions
  - `sed -n -e "/regex/p" filename`
- **awk**
  - Execute arbitrary commands for each line
  - Columns in parameters $1 $2 ...
  - `awk '{ commands }' (e.g. if, printf)`
Exercise

- Exercise 3.1
- Exercise 3.2
- Exercise 3.3

Multipoint Links (LANs)

- **LAN (multi-access)**
  - Shared medium
  - Concurrent access
  
  ```
  $ns make-lan "$n1 $n2" $bw $delay
      LL Queue/DropTail MAC/802_3 Channel
  ```

- **Creates LanNode**
  - Connects the nodes
  - Is a “Node” only from routing perspective
  - Implicitly creates a LanIface per node
  - Implicitly creates a Vlink to all nodes
Multipoint Links (LANs)

- **Channel class**
  - Simulates shared medium
  - Allows for MAC layer Carrier Sense and Collision Detection
  - `instvar delay`

- **MacClassifier Klasse**
  - Delivers Packets, resp.
  - Simple Broadcast mechanism
Multipoint Links (LANs)

Mac class

- Simulates medium access protocol
- instvar bandwidth_ modulation rate
- instvar hlen_ header length
- instvar label_ MAC address

Mac

Mac/802_3 Mac/802_11 Mac/Csma (out of date!)

Multipoint Links (LANs)

LL class (Link Layer)

- Determines the IP address of next hop
  - Uses LanRouter Object
- Determines related MAC address
  - Simplification: 1:1 of IP to MAC address (i.e. IP address is used as MAC address!)

LAN related OTcl files have own folder

- ~ns/tcl/lan/*
- Includes also LAN extension of simulator
Multipoint Links (LANs)

- **LanRouter class**
  - One object per LAN
  - Created with LanNode
  - Each link layer object stores pointer to LanRouter
  - Determines next hop in the LAN
  - Uses configured RouteLogic

Protocols in ns-2

- Validated! (~ns/tcl/test/*)
- IP
- UDP
- TCP
  - Taheo, Reno, New Reno, Vegas, ...
  - TFRC
- RTP/RTCP (Realtime Transport P.)
- SRM (Scalable Reliable Multicast)
- SCTP (Stream Control Transmission P.)
Traffic profiles in ns-2

- Out-of-box
- CBR (Constant Bit Rate)
- Exponential
- Pareto
- RealAudio
- Worm (Simple Worm model)
- ScpApp
- Web cache
- Web traffic

Queuing discipline

- DropTail (simple FIFO)
- RED (Random Early Detect)
  - PD
  - dsRED (RED for DiffServ)
- DRR (Deficit Round-Robin)
- SFQ (Stochastically Fair Queuing)
- CBQ (Class based Queuing)
- XCP (Explicit Congestion)
Exercise

- Exercise 4.1
- Exercise 4.2

Unicast Routing

- No protocols, only algorithms are implemented
  - `$ns rtproto Protocol Nodelist`
- Static (Standard)
  - Dijkstra All-pairs Shortest Path First (SPF)
  - Calculated once before start of simulation
- Session
  - also Dijkstra (as static)
  - Recalculation for each topology change
  - Central routing protocol
Unicast Routing

- **DV**
  - Dynamic, decentralized, not for LANs!
  - Creates an agent per node
  - Distance Vector (Distributed Bellman-Ford)
  - Supports link costs
    \[ \text{costs} \]
    \[ \text{costs} \]
  - Supports multiple paths to the same destination
    \[ \text{set multiPath_} \]

- **Manual**
  - No protocol
  - equivalent to "route" command

Network Dynamics

- **Activate and deactivate**
  - Links
  - Nodes

- **Patterns**
  - Manual (once)
    \[ \text{rtmodel} \]
    \[ \text{rtmodel} \]

  - operation: link-up, link-down
    - Two nodes: Link in between
    - One node: all links to this node

\[ \text{set multiPath_} \]

\[ \text{at} \]

\[ \text{up|down} \]

\[ \text{n1 \ n2} \]
Network Dynamics

– Exponential distribution
$ns rtmodel Exponential \{start uptime downtime finish\} n1 \{n2\}

– Deterministic
$ns rtmodel Deterministic \{start uptime downtime finish\} n1 \{n2\}

– From $start sec for uptime activated, then for downtime deactivated, until $finish sec.
– uptime and downtime for exponential documented wrong!

Network Dynamics

– Timestamps in file
$ns rtmodel Trace filename n1 \{n2\}
   • File format
      v time operation n1 \{n2\}
      • Operation: see above
      • Not valid lines (e.g., different nodes) are ignored

 Recording of events/actions
$rtmodel trace-dynamics $file
– Writes above format
Hierarchical Routing

- More efficient routing tables
- If number of nodes exceeds several thousands

3 layers possible:

Layer 1: domain
Layer 2: cluster
Layer 3: node

Exercise

- Exercise 5.1
- Exercise 5.2
- Exercise 5.3
Bug in ns-2.3x

- On activating or deactivating of a link:
  
  ```
  foreach node [[Simulator instance] all-nodes-list]
  {
      # XXX using dummy 0 for 'changes'
      $node notify-mcast 0
  }
  
  - “LAN” is a node (LanNode) but misses notify-mcast
  - Hack: find method notify-mcast of node
  - Create an (empty) method stub for LanNode
  - Run make in folder ns-2.33
  ```

Outlook

- **Error Model**
- **Wireless**
  - 802.11 MAC
  - Movement model
  - Ad Hoc Routing
  - Antenna
  - Mobil IP
  - Energy consumption
  - Satellite networks
Outlook

- **Emulation**
  - Connects simulator to real world networks
- **Topology Generators**
- **Random Variables**

Further documentation

- **nsnam Homepage**
  - User Information
- **Marc Greis’s tutorial**
  - Good starting point
- **Ns by Example**
  - Some more examples
- **Ns Manual**
  - The ns-2 bible
  - Reference for development
Further documentation

- **Slides of other Tutorials**
  - ISI ns tutorial 2002
  - UCB ns workshop 1999

- **Internet research**
  - Astonishing less helpful

- **Source code**
  - In your own installation directory
  - Doxygen formatted
    - Search for “doxygen ns-2”
  - http://www-sop.inria.fr/planete/software/

ns-2 Internals

- **Size** (Yu., H., Salehi, N., IEC Workshop 2000)
  - 100,000 lines of C++-Code
  - 70,000 lines of OTcl-Code
  - 30,000 lines of Test-Suite
  - 20,000 lines Documentation

- **Fundamental model**
  - „Split object“
  - Binding between C++ and OTcl