Introduction to Network Simulator

Giovanni NEGLIA and Mouhamad IBRAHIM
gneglia@sophia.inria.fr, mibrahim@sophia.inria.fr

www-sop.inria.fr/maestro/personnel/Giovanni.Neglia/ns_course/ns_course.htm

Maestro team

INRIA Sophia-Antipolis - France

Course objectifs

- To learn the basic simulated objects of NS, to become familiar with these objects, and to use NS to analyse some of the simulated objects.
- Configure, run and analyse simulations for wireless networks.
- Configure, run and analyse simulations for interacting TCP flows.
- A flavour of how to add new protocols to NS-2.
- A flavour of general issues in simulations.

Outline

- What is NS, what is used for.
- Install NS-2.
- Basic OTcl programming.
- Configure and run a basic simulation in NS.

Overview

- Network Simulator version 2 (NS-2) is a free and open source discrete event network simulator developed at UC Berkeley
  - You can add your own protocol, contribute to the code and, from time to time, you need to troubleshoot some of the bugs
  - NS is a discrete event simulator where the advance of time depends on the timing of events which are maintained by a scheduler.
- NS-2 works under Linux, Mac, and Windows.
- Current release is ns-2.31.
- Release under work: ns-3 where Inria takes part of the development process.

Overview (cont'd)

- NS-2 has a large and rich library of network and protocol objects.
- It covers a large part of applications (Web, FTP, CBR,. . .), protocols (transport and routing protocols), network types (Satellite links, wired and wireless LAN), network elements (mobile nodes, wireless channel models, link and queue models,. . .) and traffic models (exponential, uniform, . . .).
- NS also allows to add and test new protocols and applications and/or to modify existing ones.

Overview (cont'd)

- NS-2 is based on an object oriented simulator written in C++ and a OTcl interpreter (an object oriented extension of Tool Command Language TCL)
- These different objects are written in C++ code in order to achieve efficiency in the simulation and faster execution times. (e.g. Implementation of IEEE 802.11 is found in . . . ns-**/ns-**/mac/802_11.cc,h)
- The OTcl script, which is provided by the user at the simulation time, is used to define and to configure the network topology and network elements (node type, the protocols and applications to be used), and to schedule the events.

Overview (cont'd)

- The OTcl scripts are used also to tell NS to create a visulation trace as well as an ascii file trace corresponding to the events generated in the network.
- To analyse the trace files, other independent tools will be needed to filter, compute and display the results (e.g. Awk, Matlab, gnuplot, etc..).
- Two other independent and optional tools are provided with NS packages: Network animator (Nam) and xgraph.
- OTcl scripts can be written in any text editor like ***kate*** or ***emacs***.

Installing NS

- There are two ways to install NS: either independently by pieces or all at once (the recommended one)
- Install all pieces at once using ns-allinonexxx package. For details see: http://www.isi.edu/nsnam/ns/ns-build.html
- Once the installation passed correctly, we need to configure three environment variables. Usually, we need to change the following lines in .bash_profile file or in the corresponding file at your system:

  export PATH=$PATH:. . .
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:. . .
  export TCL_LIBRARY=$TCL_LIBRARY:. . .

NS-2 documentation

- NS-2 homepage for the code as well for the documentations: *http://www.isi.edu/nsnam/ns/* or in Wikipedia format *http://nsnam.isi.edu/nsnam/index.php*
- NS-2 references and tutorials:
  - NS Manual also called NS notes: *http://www.isi.edu/nsnam/ns/ns-documentation.html*
  - NS for beginners by E. Altman and T. Jimenez: *http://www.sop-inria.fr/mistral/personnel/Eitan.Altman/COURS-NS/n3.pdf*
  - Marc Greis Tutorial: *http://www.isi.edu/nsnam/ns/tutorial/*
  - NS by Example Jae Chung Mark Claypool: *http://nile.wpi.edu/NS/*
- NS-2 examples: you can find it at the following directory: *ns-allinone-**/ns-**/tcl/ex/*

OTcl programming

- OTcl is an object oriented extension of the Tool Command Langage TCL.
- Tcl is a free script language that has a simple syntax, easy to be integrate with other languages and plateform independent.
- Basic instructions:
  - Set a value to a variable: `set a 1`
    note in Tcl, variables are not typed. The value assigned to a variable determines the type of the variable.
  - To use the value assigned to a variable, we need to use the $ sign: `set x $a`
  - A mathematical operation is done using the expression command `expr` as follows:
    ```
    set b [expr $x +-*/% $a]
    ```

OTcl programming (cont'd)

- To comment a line, use the symbol `#`
- To create a file, we need to give it a name that will identify it by our system. Then we need to assign a pointer to it that will be used within the Tcl program in order to relate it:

```
set tclFilePointer [open ourFile.txt w]
```

- To print an output, use the command `puts " "`. For instance, to print into `ourFile.txt` the previous variable b with its tabulated value, use the following line:

```
puts $tclFilePointer "b \t $b"
```

- To execute a Linux command from the Tcl script, use the command `exec`. For example, to plot the curve of the data contained in `ourFile.txt` e.g., use:

```
exec gnuplot ourFile.txt &
```

& is to allow the command `gnuplot` to execute in the background.

OTcl programming (cont'd)

- An `if` command has the following structure:

```
if { expression } {
  Tcl commands
} else {
  Tcl commands }
```

  - We can nest several `if` and `else` commands in the `if/else` part as in `C` language.
  - Condition equality is tested using == and inequality using `!=`
  - `For` loop is declared as follows:

```
for {set i 0} {$i < 6} {incr i 2} {
  execute some commands }
```

  - We can define a table `tab` in Tcl as follows:

```
for {set i 1} {$i <= $tableSize} {incr i} {
 set tab($i) $value }
```

OTcl programming (cont'd)

- We can also declare a procedure/function in Tcl. The syntax is as follows:

```
proc procedureName {parameter1 parameter 2 ...}
{
    global var1 var2 ...
    Tcl commands
    return $something }
```

- The procedure is called by typing:

```
procedureName parameter1 parameter 2 ...
```

- Reserved word `global` is generally used to change external variables to the procedure.

OTcl programming (cont'd)

- To instantiate an object of a class:
  ```
  set obj [new Class1/Class2 ]
  ```
- To assign a value to an object attribute:
  ```
  $obj set attribute1 $x
  ```
- To read the value of an object attribute:
  ```
  set a [ $obj set attribute2 ]
  ```
- To call the method without a return value of object `obj`:
  ```
  $obj method par1 par2 ...
  ```
- To call the method with a return value of object `obj`:
  ```
  set a [ $obj method par1 par2 ...]
  ```

- Write a procedure/function that returns all the prime numbers of a certain value x.

Steps of a NS simulation

- Define the scenario to simulate:
  1. Create the simulator object
  2. { Turn on tracing }
  3. Setup the network nodes {and links }
  4. Setup the routing mechanism
  5. Create transport connections
  6. Setup user applications
  7. Schedule data transmission
  8. Stop the simulation
- Execute the OTcl script in a Linux shell: $>$ ns example.tcl
- Extract the results from the trace files: `awk`, `xgraph`, `nam`, `matlab`, etc ...

Creating a Tcl scenario

- NS simulation starts with the command:

```
set ns [new Simulator]
```

- ns is now an instance of the class `Simulator`, so we can use its methods and its attributes.

- To define trace files with the data that needs to be collected from the simulation, we have to create these files using the command open:

```
# open the trace file
set traceFile [open out.tr w]
$ns trace-all $traceFile
# open the Nam trace file
set namFile [open out.nam w]
$ns namtrace-all $namFile
```

Creating a Tcl scenario (cont'd)

- To schedule an event:

```
$ns at time "event"
```

- To terminate a Tcl scenario, we need to call the `exit` command:

```
$ns at 12.0 "exit 0"
```

- Usually, we declare a `finish` procedure to dump the traces and to close the files:

```
proc finish {} {
  global ns traceFile namFile
  $ns flush-trace
  close $traceFile
  close $namFile
  exec nam out.nam &
  exit 0 }
```

Creating a Tcl scenario (cont'd)

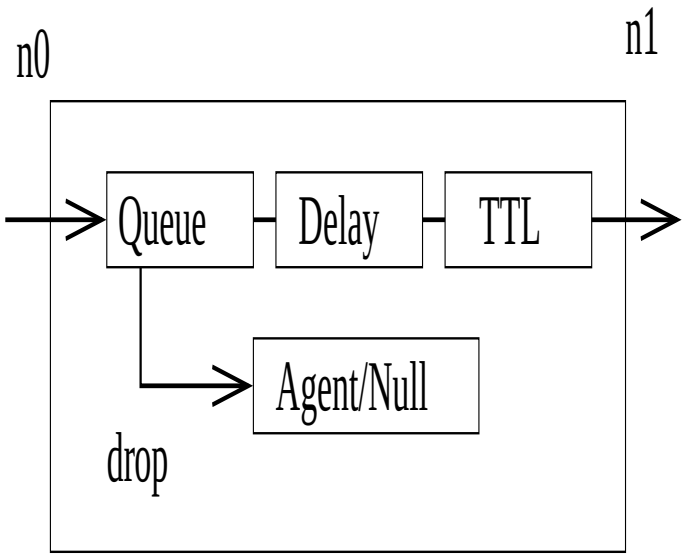
- Similarly, we need to declare explicitly to NS when to call the `finish` function in order to stop the simulation:
  ```
  $ns at 12.0 "finish"
  ```
- The simulation can then begin with the last command in the Tcl script: `$ns run`
- Creation of nodes and links in NS:
  - To create a node: `set n0 [$ns node]`
  - To create a given number `nodeNb` of nodes:
    ```
    for {set i 1} {$i <= $nodeNb} {incr i} {
     set n($i) [$ns node] }
    ```

Creating a Tcl scenario (cont'd)

- To create a link between two nodes, we need to specify the parameters of the link:

```
$ns simplex-link/duplex-link $n0 $n1
"bandwidth"Mb "delay"ms "queue_type"
```

- A duplex link in NS is represented as two parallel simplex-links.
- A simplex-link has the following representation in NS:



Creating a Tcl scenario (cont'd)

- Output queue of a node is represented as input queue of the corresponding link.
- There are different queue types: Drop tail, Stochastic Fair queueing (SFQ), . . .
- Packet overflow is implemented by sending dropped packets to the Null agent.
- TTL object computes the Time To Live for each received packet.
- To set the queue size of the ingress nodes of a link to some limit:
  `$ns queue-limit $n0 $n1 20`
- By default, the queue limit is set to 50. All the default values can be found and modified at ***. . . /ns-**/tcl/lib/ns-default.tcl***

Creating a Tcl scenario (cont'd)

- Next step is to create transport agents, to attach them to corresponding nodes, and to associate them to each other. We need also to configure their different parameters.
- For a TCP agent:

```
set tcp [new Agent/TCP]
$ns attach-agent $n(0) $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n(1) $sink
$ns connect $tcp $sink
```

- For a UDP agent:

```
set udp [new Agent/UDP]
$ns attach-agent $n(0) $udp
set null [new Agent/Null]
$ns attach-agent $n(1) $null
$ns connect $udp $null
```

Creating a Tcl scenario (cont'd)

- Next, we need to create the applications, to attach them to corresponding transport agents, and to configure their parameters:
- To setup a FTP application e.g., we need a TCP transport agent:

```
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

- To setup a CBR application e.g., we need a UDP transport agent:

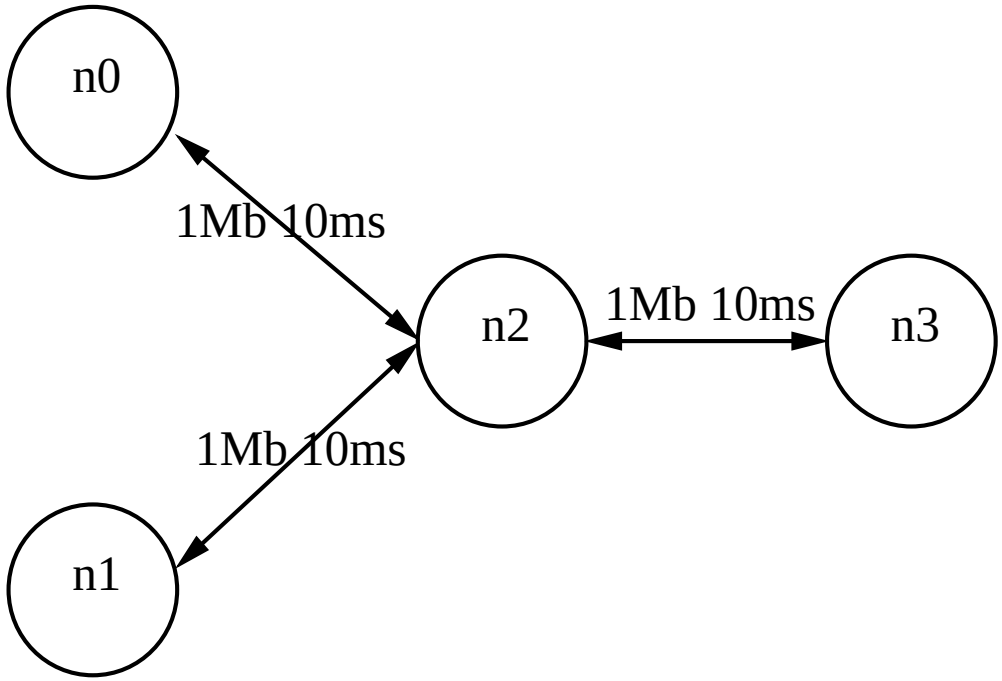
```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
```

- Before calling the `finish` procedure, we need to schedule the `start time` and the `stop time` for each data sources:

```
$ns at 10.0 "$cbr start"
$ns at 20.0 "$cbr stop"
```

Simulation scenario sample1

- Write the Tcl script for the following network shown below. There is a CBR application running over a UDP protocol at node n0 and sending packets of 512 bytes to node n3 at the rate of 0.005. There is also another CBR application running over a UDP protocol at node n1 and sending packets with the same size and rate to node n3. Queue type is Drop tail.



Simulation scenario sample2

- Write the Tcl script for the following network shown below. There is an FTP application running at node n0 and sending traffic to node n4. Packets have size of 512 bytes. There is a CBR application running over a UDP protocol at node n1 and sending traffic to node n5 at the rate of 0.01 Mb. Packets have size of 1000 bytes.