# Lab 1 NS2 Warm Up

March 4, 2011

## Contents

**Title:** **1**

**1 The Basics of NS2** **2**

1.1 Downloading/Installing ns & nam . . . . . . . . . . . . . . . . . . . . . . . 2

1.2 Starting NS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.3 Starting nam . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

**2 The First Tcl Script** **4**

2.1 How to start . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.2 A Simple Script of 2 nodes, 1 link . . . . . . . . . . . . . . . . . . . . . . 4

2.3 Sending data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

**3 Making it more interesting** **6**

3.1 The topology . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.2 The events . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3.3 Marking flows . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

3.4 Monitoring a queue . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

**4 Creating Output Files for Xgraph** **10**

4.1 Topology and Traffic Sources . . . . . . . . . . . . . . . . . . . . . . . . 11

4.2 Recording Data in Output Files . . . . . . . . . . . . . . . . . . . . . . . 12

4.3 Running the Simulation . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

**References** **15**

**A NS2 Setup** **16**

**B The Source Code** **18**

# 1 The Basics of NS2

Ns or the network simulator (also popularly called ns-2, in reference to its current generation) is a discrete event network simulator. Ns is popularly used in the simulation of routing and multicast protocols, among others, and is heavily used in ad-hoc networking research. Ns supports an array of popular network protocols, offering simulation results for wired and wireless networks alike. It can be also used as limited-functionality network emulator. It is popular in academia for its extensibility (due to its open source model) and plentiful online documentation. NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkely written in C++ and OTcl, which is very useful for simulatry local and wide area networks. Although NS is fairly easy to use once you get to know the simulator, it is quite difficult for a first time user, because there are few user-friendly manuals.

The purpose of this Lab is to get some basic idea of how the simultor works, how to setup simulation networks, where to look for further information about network components in simulator codes, how to create new network components, etc., mainly by doing several simple examples and reading brief explanations.

The best website for getting started would be
`http://www.isi.edu/nsnam/ns/ns-build.html`.

## 1.1 Downloading/Installing ns & nam

You can build ns either from the the various packages (Tcl/Tk, otcl, etc.), or you can download an 'all-in-one' package. Highly recommend you to start with the all-in-one package, especially if you're not entirely sure which packages are installed on your system, and where exactly they are installed. Note: The all-in-one package only works on Linux systems. More details for installation would be in the appendix.

## 1.2 Starting NS

Start ns with the command 'ns <tcl script>' (assuming that you are in the directory with the ns executable, or that your path points to that directory), where '<tcl script>' is the name of a Tcl script file which defines the simulation scenario (i.e. the topology and the events). You could also just start ns without any arguments and enter the Tcl commands in the Tcl shell, but that is definitely less comfortable.

Everything else depends on the Tcl script. The script could create some output on stdout, it might write a trace file or it might start nam to visualize the simulation.

## 1.3 Starting nam

You can either start nam with the command 'nam <nam-file>', where '<nam-file>' is the name of a nam trace file that was generated by ns, or you can execute it directly out of the Tcl simulation script for the simulation which you want to visualize. For additional parameters to nam, see the nam manual page. Figure 1 shows a screenshot of a nam window where the most important functions are being explained.

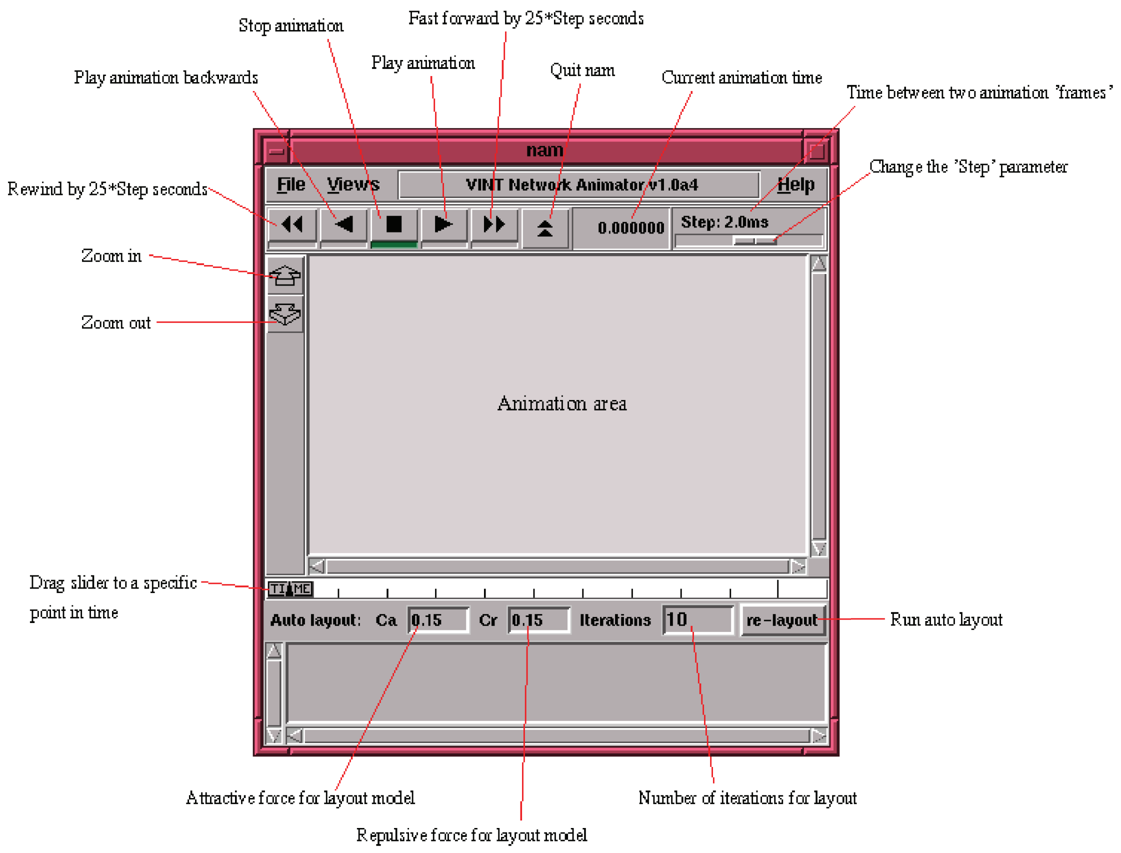

Stop animation
Fast forward by 25*Step seconds
Play animation
Quit nam
Play animation backwards
Current animation time
Time between two animation 'frames'
nam
Change the 'Step' parameter
Rewind by 25*Step seconds
File
Views
VINT Network Animator v1.0a4
Help
0.000000
Step: 2.0ms
Zoom in
Zoom out
Animation area
Drag slider to a specific point in time
TIME
Auto layout:
Ca
0.15
Cr
0.15
Iterations
10
re-layout
Run auto layout
Attractive force for layout model
Number of iterations for layout
Repulsive force for layout model

Figure 1: Nam window sample

# 2 The First Tcl Script

In this section, we are going to develop a Tcl script for ns which simulates a simple topology. we are going to learn how to set up nodes and links, how to send data from one node to another, how to monitor a queue and how to start nam from simulation script to visualize simulation.

## 2.1 How to start

Now we are going to write a ’template’ that you can use for all of the first Tcl scripts. You can write your Tcl scripts in any text editor as you wish. I suggest that you call this first example ’example1.tcl’.

First of all, you need to create a simulator object. This is done with the command:

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

Now we open a file for writing that is going to be used for the nam trace data.

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

The first line opens the file ’out.nam’ for writing and gives it the file handle ‘nf’. In the second line we tell the simulator object that we created above to write all simulation data that is going to be relevant for nam into this file.

The next step is to add a ’finish’ procedure that closes the trace file and starts nam.

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

The next line tells the simulator object to execute the ’finish’ procedure after 5.0 seconds of simulation time.

```
$ns at 5.0 "finish"
```

You probably understand what this line does just by looking at it. ns provides you with a very simple way to schedule events with the ’at’ command.

The last line finally starts the simulation.

```
$ns run
```

You will have to use the code from this section as starting point in the other sections.

## 2.2 A Simple Script of 2 nodes, 1 link

In this section we are going to define a very simple topology with two nodes that are connected by a link. The following two lines define the two nodes. (Note: You have to insert the code in this section before the line ’$nsrun', orevenbetter, beforetheline'$ns at 5.0 ”finish”’).

```
set n0 [$ns node]
set n1 [$ns node]
```

A new node object is created with the command ’$ns node’. The above code creates two nodes and assigns them to the handles ’n0’ and ’n1’.

The next line connects the two nodes.

```
$ns duplex−link $n0 $n1 1Mb 10ms DropTail
```

This line tells the simulator object to connect the nodes n0 and n1 with a duplex link with the bandwidth 1Megabit, a delay of 10ms and a DropTail queue.

Now you can save your file and start the script with ’ns example1.tcl’. nam will be started automatically and you should see an output that resembles in figure 2.

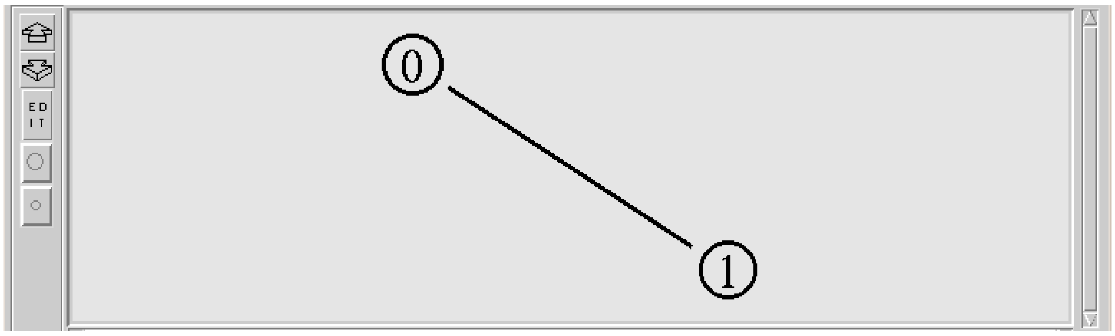


Figure 2: Example 1-1

## 2.3 Sending data

Of course, this example isn’t very satisfying yet, since you can only look at the topology, but nothing actually happens, so the next step is to send some data from node n0 to node n1. In ns, data is always being sent from one ’agent’ to another. So the next step is to create an agent object that sends data from node n0, and another agent object that receives the data on node n1.

```
# Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach−agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Applicaiton/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set inteval_  0.005
$cbr0 attach−agent $udp0
```

These lines create a UDP agent and attach it to the node n0, then attach a CBR traffic generater to the UDP agent. CBR stands for ’constant bit rate’. Line 7 and 8 should be self-explaining. The packetSize is being set to 500 bytes and a packet will be sent every 0.005 seconds (i.e. 200 packets per second). You can find the relevant parameters for each agent type in the ns manual page.

The next lines create a Null agent which acts as traffic sink and attach it to node n1.

```
set null0 [new Agent/Null]
$ns attach−agent $n1 $null0
```

Now the two agents have to be connected with each other.

```
$ns conncet $udp0 $null0
```

And now we have to tell the CBR agent when to send data and when to stop sending. Note: It's probably best to put the following lines just before the line '$ns at 5.0 "finish"'.

```
$ns at 0.5 "$cbr0 start"
$ns at 4.0 "$cbr0 stop"
```

This code should be self-explaining again.

Now you can save the file and start the simulation again. When you click on the 'play' button in the nam window, you will see that after 0.5 simulation seconds, node 0 starts sending data packets to node 1. figure 3 You might want to slow nam down then with the 'Step' slider.

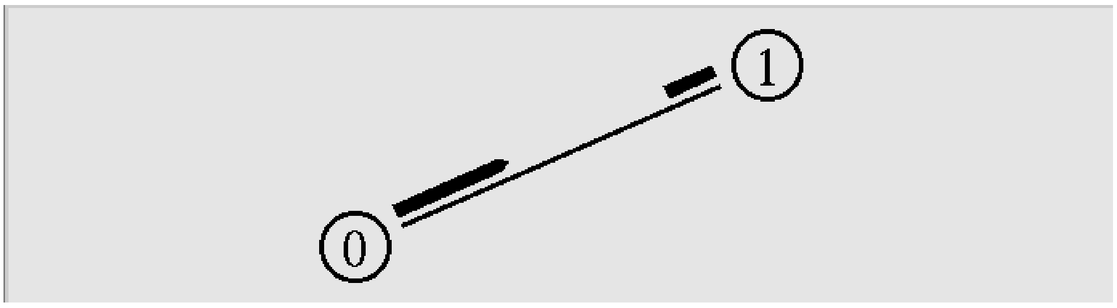


Figure 3: Example 1-2

Strongly suggest that now you start some experiments with nam and the Tcl script. You can click on any packet in the nam window to monitor it, and you can also click directly on the link to get some graphs with statistics. Also suggest that you try to change the 'packetsize_' and 'interval_' parameters in the Tcl script to see what happens.

Most of the information that needed to be able to write this Tcl script was taken directly from the example files in the 'tcl/ex/' directory, while you learned which CBR agent arguments (packetSize_, interval_) you had to set from the ns manual page.

## 3 Making it more interesting

In this section we are going to define a topology with four nodes in which one node acts as router that forwards the data that two nodes are sending to the fourth node. This section will explain find a way to distinguish the data flows from the two nodes from each other, and this section will also show how a queue can be monitored to see how full it is, and how many packets are being discarded.

## 3.1 The topology

As always, the first step is to define the topology. You should create a file 'example2.tcl', using the code from section above as a template. As it has been mentioned before, this code will always be similar. You will always have to create a simulator object, you will always have to start the simulation with the same command, and if you want to run nam automatically, you will always have to open a trace file, initialize it, and define a procedure which closes it and starts nam.

Now insert the following lines into the code to create four nodes.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

And connect them

```
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail
```

You can save and start the script now. You might notice that the topology looks a bit awkward in nam. You can hit the 're-layout' button to make it look better, but it would be nice to have some more control over the layout. Add the next three lines to your Tcl script and start it again.

```
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```

You will probably understand what this code does when you look at the topology in the nam window now. It should look like in figure 4.

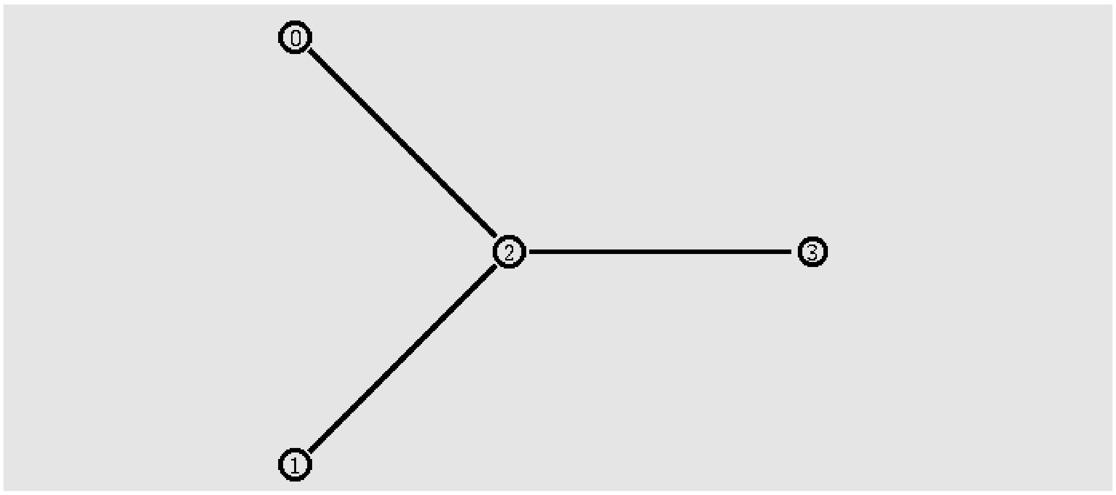


Figure 4: Example 2-1

Note that the autolayout related parts of nam are gone, since now you have taken the layout into your own hands. The options for the orientation of a link are right, left, up, down and combinations of these orientations. You can experiment with these settings later, but for now please leave the topology the way it is.

## 3.2 The events

Now we create two UDP agents with CBR traffic sources and attach them to the nodes n0 and n1. Then we create a Null agent and attach it to node n3.

```
# Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

# Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

The two CBR agents have to be connected to the Null agent.

```
$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

We want the first CBR agent to start sending at 0.5 seconds and to stop at 4.5 seconds while the second CBR agent starts at 1.0 seconds and stops at 4.0 seconds.

```
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
```

When you start the script now with 'ns example2.tcl', you will notice that there is more traffic on the links from n0 to n2 and n1 to n2 than the link from n2 to n3 can carry. A simple calculation confirms this: We are sending 200 packets per second on each of the first two links and the packet size is 500 bytes. This results in a bandwidth of 0.8 megabits per second for the links from n0 to n2 and from n1 to n2. That's a total bandwidth of 1.6Mb/s, but the link between n2 and n3 only has a capacity of 1Mb/s, so obviously some packets are being discarded. But which ones? Both flows are black, so the only way to find out what is happening to the packets is to monitor them in nam by clicking on them. In the next two sections I'm going to show you how to distinguish between different flows and how to see what is actually going on in the queue at the link from n2 to n3.

## 3.3 Marking flows

Add the following two lines to your CBR agent definitions.

```
$udp0 set fid_ 1
$udp1 set fid_ 2
```

The parameter 'fid_' stands for 'flow id'.

Now add the following piece of code to your Tcl script, preferably at the beginning after the simulator object has been created, since this is a part of the simulator setup.

```
$ns color 1 Blue
$ns color 2 Red
```

This code allows you to set different colors for each flow id. figure 5

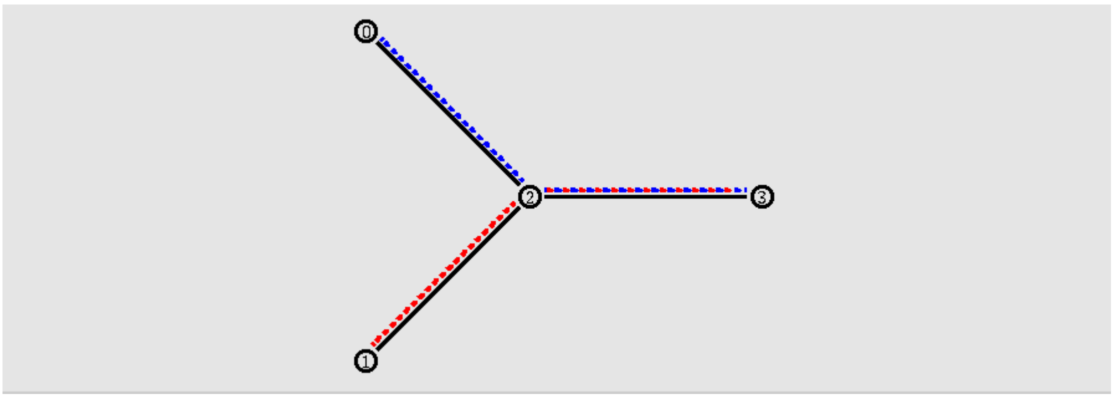


Figure 5: Example 2-2

Now you can start the script again and one flow should be blue, while the other one is red. Watch the link from node n2 to n3 for a while, and you will notice that after some time the distribution between blue and red packets isn't too fair anymore (at least that's the way it is on my system). In the next section I'll show you how you can look inside this link's queue to find out what is going on there.

## 3.4 Monitoring a queue

You only have to add the following line to your code to monitor the queue for the link from n2 to n3.

```
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

Start ns again and you will see a picture similar to figure 6 after a few moments.

You can see the packets in the queue now, and after a while you can even see how the packets are being dropped, though (at least on my system, I guess it might be different in later or earlier releases) only blue packets are being dropped. But you can't really expect too much 'fairness' from a simple DropTail queue. So let's try to improve the queueing by using a SFQ (stochastic fair queueing) queue for the link from n2 to n3. Change the link definition for the link between n2 and n3 to the following line.

```
$ns duplex-link $n3 $n2 1Mb 10ms SFQ
```

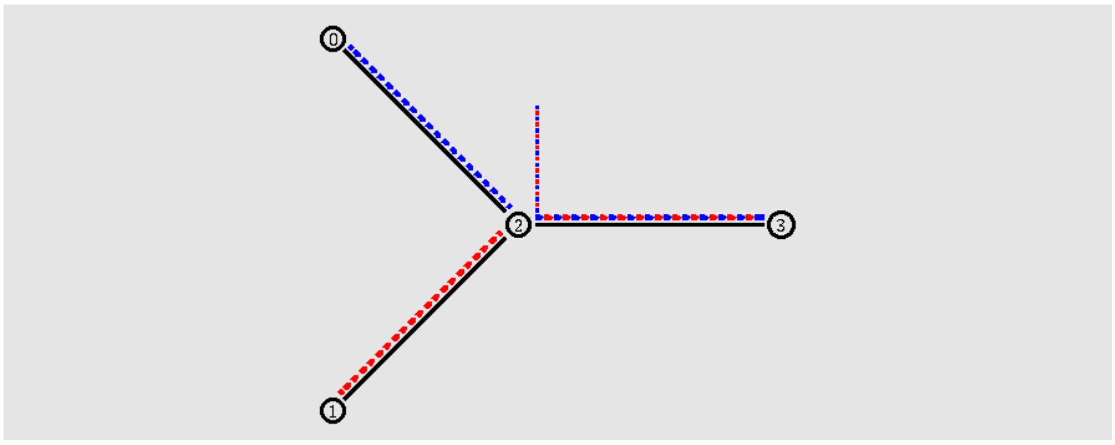


Figure 6: Example 2-3

The queueing should be 'fair' now. The same amount of blue and red packets should be dropped. figure 7

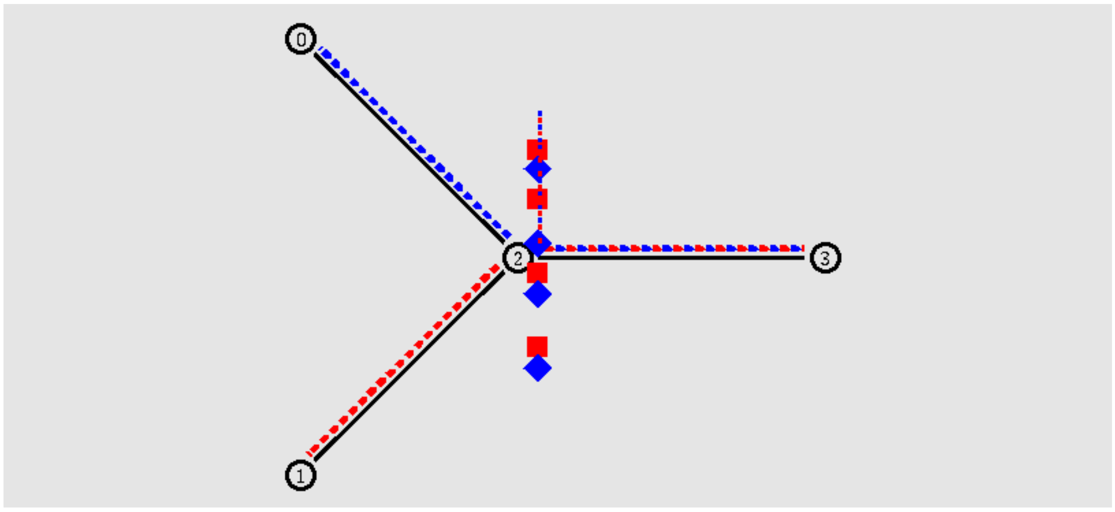


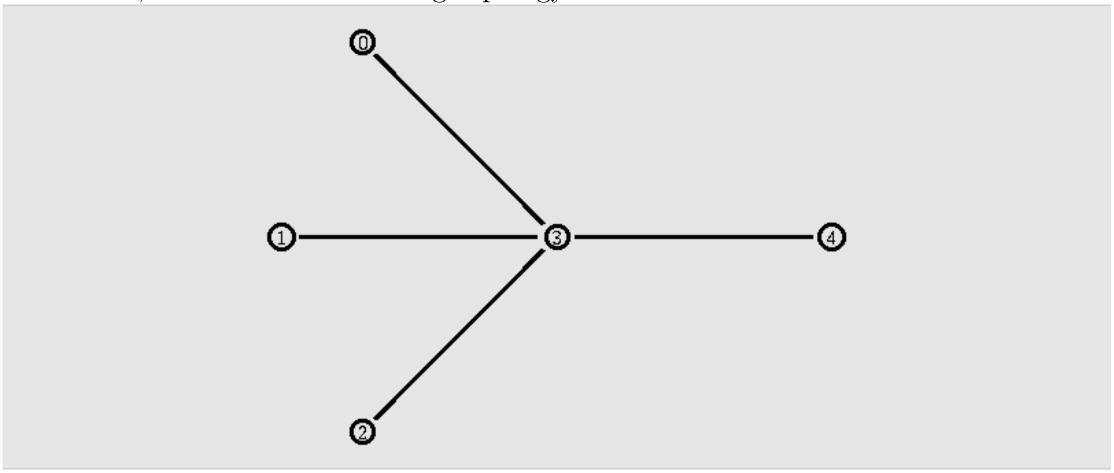
Figure 7: Example 2-4

## 4 Creating Output Files for Xgraph

One part of the ns-allinone package is 'xgraph', a plotting program which can be used to create graphic representations of simulation results. In this section, I will show you a simple way how you can create output files in your Tcl scripts which can be used as data sets for xgraph. On the way there, I will also show you how to use traffic generators.

## 4.1 Topology and Traffic Sources

First of all, we create the following topology:



The following piece of code should look familiar to you by now if you read the first sections of this tutorial.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

$ns duplex-link $n0 $n3 1Mb 100ms DropTail
$ns duplex-link $n1 $n3 1Mb 100ms DropTail
$ns duplex-link $n2 $n3 1Mb 100ms DropTail
$ns duplex-link $n3 $n4 1Mb 100ms DropTail
```

We are going to attach traffic sources to the nodes n0, n1 and n2, but first we write a procedure that will make it easier for us to add the traffic sources and generators to the nodes:

```
proc attach-expoo-traffic { node sink size burst idle rate } {
    #Get an instance of the simulator
    set ns [Simulator instance]

    #Create a UDP agent and attach it to the node
    set source [new Agent/UDP]
    $ns attach-agent $node $source

    #Create an Expoo traffic agent and set its configuration parameters
    set traffic [new Application/Traffic/Exponential]
    $traffic set packetSize_ $size
    $traffic set burst_time_ $burst
    $traffic set idle_time_ $idle
    $traffic set rate_ $rate

    # Attach traffic source to the traffic generator
    $traffic attach-agent $source

    #Connect the source and the sink
    $ns connect $source $sink
    return $traffic
}
```

This procedure looks more complicated than it really is. It takes six arguments: A node, a previously created traffic sink, the packet size for the traffic source, the burst and idle times (for the exponential distribution) and the peak rate. For details about the Expoo traffic sources, please refer to the documentation for ns.

First, the procedure creates a traffic source and attaches it to the node, then it creates a Traffic/Expoo object, sets its configuration parameters and attaches it to the traffic source, before eventually the source and the sink are connected. Finally, the procedure returns a handle for the traffic source. This procedure is a good example how reoccuring tasks like attaching a traffic source to several nodes can be handled. Now we use the procedure to attach traffic sources with different peak rates to n0, n1 and n2 and to connect them to three traffic sinks on n4 which have to be created first:

```
set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
$ns attach-agent $n4 $sink0
$ns attach-agent $n4 $sink1
$ns attach-agent $n4 $sink2

set source0 [attach-expoo-traffic $n0 $sink0 200 2s 1s 100k]
set source1 [attach-expoo-traffic $n1 $sink1 200 2s 1s 200k]
set source2 [attach-expoo-traffic $n2 $sink2 200 2s 1s 300k]
```

In this example we use Agent/LossMonitor objects as traffic sinks, since they store the amount of bytes received, which can be used to calculate the bandwidth.

## 4.2 Recording Data in Output Files

Now we have to open three output files. The following lines have to appear 'early' in the Tcl script.

```
set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]
```

These files have to be closed at some point. We use a modified 'finish' procedure to do that.

```
proc finish {} {
        global f0 f1 f2
        #Close the output files
        close $f0
        close $f1
        close $f2
        #Call xgraph to display the results
        exec xgraph out0.tr out1.tr out2.tr -geometry 800x400 &
        exit 0
}
```

It not only closes the output files, but also calls xgraph to display the results. You may want to adapt the window size (800x400) to your screen size.

Now we can write the procedure which actually writes the data to the output files.

```
proc record {} {
        global sink0 sink1 sink2 f0 f1 f2
        #Get an instance of the simulator
        set ns [Simulator instance]
        #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 [$sink0 set bytes_]
        set bw1 [$sink1 set bytes_]
        set bw2 [$sink2 set bytes_]
        #Get the current time
        set now [$ns now]
        #Calculate the bandwidth (in MBit/s) and write it to the files
        puts $f0 "$now [expr $bw0/$time*8/1000000]"
        puts $f1 "$now [expr $bw1/$time*8/1000000]"
        puts $f2 "$now [expr $bw2/$time*8/1000000]"
        #Reset the bytes_ values on the traffic sinks
        $sink0 set bytes_ 0
        $sink1 set bytes_ 0
        $sink2 set bytes_ 0
        #Re-schedule the procedure
        $ns at [expr $now+$time] "record"
}
```

This procedure reads the number of bytes received from the three traffic sinks. Then it calculates the bandwidth (in MBit/s) and writes it to the three output files together with the current time before it resets the bytes_ values on the traffic sinks. Then it re-schedules itself.

## 4.3 Running the Simulation

We can now schedule the following events:

```
$ns at 0.0 "record"
$ns at 10.0 "$source0 start"
$ns at 10.0 "$source1 start"
$ns at 10.0 "$source2 start"
$ns at 50.0 "$source0 stop"
$ns at 50.0 "$source1 stop"
$ns at 50.0 "$source2 stop"
$ns at 60.0 "finish"

$ns run
```

First, the 'record' procedure is called, and afterwards it will re-schedule itself periodically every 0.5 seconds. Then the three traffic sources are started at 10 seconds and stopped at 50 seconds. At 60 seconds, the 'finish' procedure is called. You can find the full example script here.

When you run the simulation, an xgraph window should open after some time which should look similar to figure 8:

As you can see, the bursts of the first flow peak at 0.1Mbit/s, the second at 0.2Mbit/s and the third at 0.3Mbit/s. Now you can try to modify the 'time' value in the 'record'

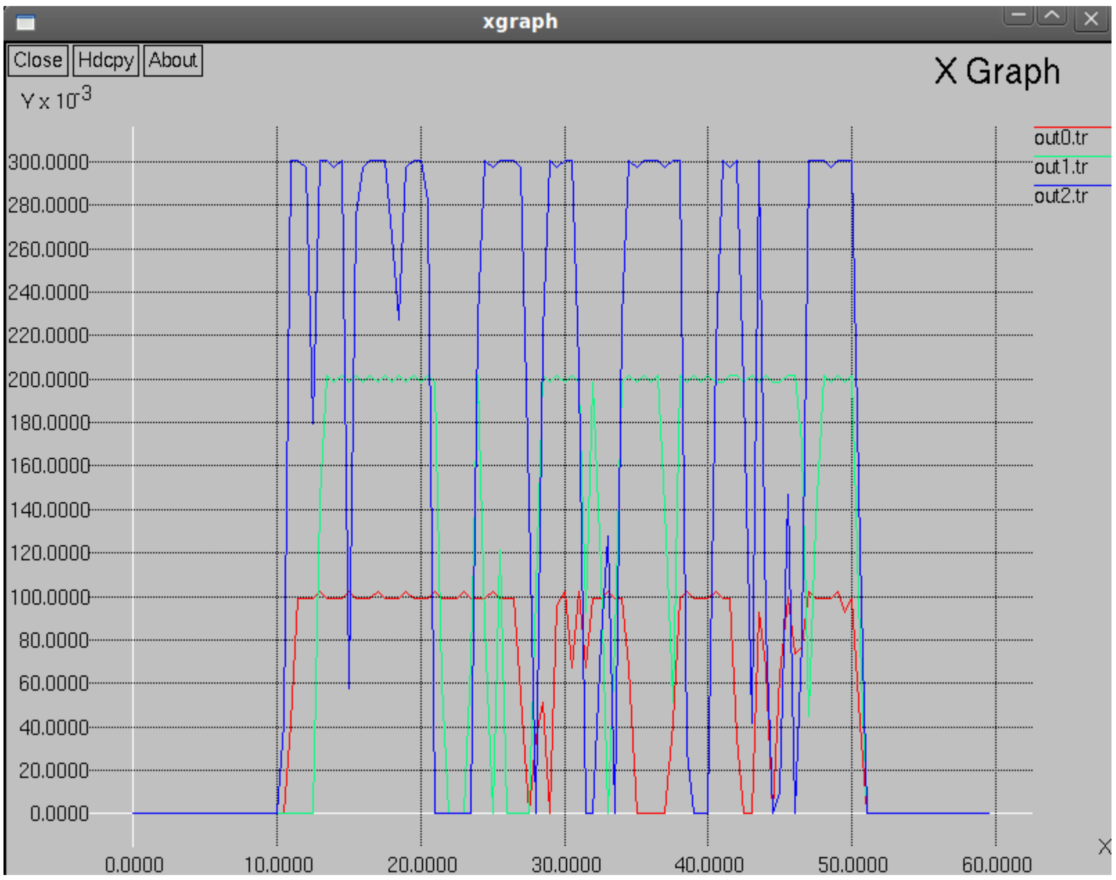


Figure 8: The xgraph

procedure. Set it to '0.1' and see what happens, and then try '1.0'. It is very important to find a good 'time' value for each simulation scenario.

Note that the output files created by the 'record' procedure can also be used with gnuplot.

## References

[1] Zhike Heng. *NS2 Tutorial by Heng*, http://140.116.72.80/ smallko/ns2/ns2.htm

[2] `http://www.isi.edu/nsnam/ns/`

[3] `http://www.answers.com/topic/ns-simulator`

[4] Jae Chung, Mark Claypool: *NS by Example*

[5] Univ. de Los Andes. Mérida. Venezuela and ESSI, Sophia-Antipolis. France: *NS Simulator for beginners*

## A NS2 Setup

See `http://nsnam.isi.edu/nsnam/index.php/User_Information`:

Step 1: Downloading ns-allinone-2.34 from
`http://downloads.sourceforge.net/nsnam/ns-allinone-2.34.tar.gz`

Step 2: Copy ns-allinone-2.34 to the path where it will be setup. We take /home/wl/-Downloads as an example(wl is the user name for the desktop).

Step 3: Set the compiling environment.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
$ sudo apt-get remove gcc  #remove old version of gcc
$ sudo apt-get install build-essential   #for gcc and some essential
$ sudo apt-get install tcl8.4 tcl8.4-dev tk8.4 tk8.4-dev   #for tcl and tk
$ sudo apt-get install libxmu-dev libxmu-headers   #for nam
```

Step 4: Modify the setup configure of ns-allinone-2.34.

```
$ cd /home/wl/Downloads/ns-allinone-2.34/otcl-1.13
$ sudo gedit configure.in
```

Change *SHLIB_LD="ld -shared"* to *SHLIB_LD="gcc -shared"* in line 77. Save and quit.

```
$ sudo gedit configure
```

Also change *SHLIB_LD="ld -shared"* to *SHLIB_LD="gcc -shared"* in line 6000+. You can use Ctrl+F to find it. Save and quit.

Step 5: The installation.

```
$ cd /home/wl/Downloads/ns-allinone-2.34
$ sudo ./install
```

Watch and wait.

Step 6: Set environment variables.

```
$ sudo gedit ~/.bashrc
```

Add the following at the end of this file(Note: wl is the user name for the desktop, you should change it to your own user name):

```
# LD_LIBRARY_PATH
OTCL_LIB=/home/wl/Downloads/ns-allinone-2.34/otcl-1.13
NS2_LIB=/home/wl/Downloads/ns-allinone-2.34/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export LD_LIBRARY_PATH = $LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:
          $X11_LIB:$USR_LOCAL_LIB

# TCL_LIBRARY
TCL_LIB=/home/wl/Downloads/ns-allinone-2.34/tcl8.4.18/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB
```

```
# PATH
XGRAPH=/home/wl/Downloads/ns−allinone−2.34/bin:/home/wl/Downloads/ns−
        allinone−2.34/tcl8.4.18/unix:/home/wl/Downloads/ns−allinone−2.34/tk8
        .4.18/unix
NS=/home/wl/Downloads/ns−allinone−2.34/ns−2.34/
NAM=/home/wl/Downloads/ns−allinone−2.34/nam−1.14/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

Save and quit. And make it work at first place:

```
$ source ~/.bashrc
```

Step 7: Test it for setup successfully.

```
$ ns
```

The "%" symbol appears on the screen. Type "exit" to quit.

Step 8: Install xgraph.

```
$ sudo apt−get install xgraph
```

## B The Source Code

```
# Create a simulator object
set ns [new Simulator]

# Define colors
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green

# Open the output files for recording
set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]

# Open a file for the nam trace data
set nf [open out.nam w]
$ns namtrace−all $nf

# Define the 'finish' procedure
proc finish {} {
    global f0 f1 f2 ns nf
    #Close the output files
    close $f0
    close $f1
    close $f2
    close $nf
    exec nam out.nam &
    #Call xgraph to display the results
    exec xgraph out0.tr out1.tr out2.tr −geometry 800x600 &
    exit 0
    }

# Define the 'record' procedure
proc record {} {
    global sink0 sink1 sink2 f0 f1 f2
    #Get an instance of the simulator
    set ns [Simulator instance]
    #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 [$sink0 set bytes_]
    set bw1 [$sink1 set bytes_]
    set bw2 [$sink2 set bytes_]
    #Get the current time
    set now [$ns now]
    #Calculate the bandwidth (in MBit/s) and write it to the files
    puts $f0 "$now␣[expr␣$bw0/$time*8/1000000]"
    puts $f1 "$now␣[expr␣$bw1/$time*8/1000000]"
    puts $f2 "$now␣[expr␣$bw2/$time*8/1000000]"
    #Reset the bytes_ values on the traffic sinks
    $sink0 set bytes_ 0
    $sink1 set bytes_ 0
    $sink2 set bytes_ 0
    #Re−schedule the procedure
    $ns at [expr $now+$time] "record"
}
```

```
# Define the attach−expoo−traffic procedure
proc attach−expoo−traffic { node sink size burst idle rate color } {
    #Get an instance of the simulator
    set ns [Simulator instance]

    #Create a UDP agent and attach it to the node
    set source [new Agent/UDP]
    $ns attach−agent $node $source
    $source set fid_ $color

    #Create an Expoo traffic agent and set its configuration parameters
    set traffic [new Application/Traffic/Exponential]
     $traffic set packetSize_ $size
     $traffic set burst_time_ $burst
     $traffic set idle_time_ $idle
     $traffic set rate_ $rate

    #Attach traffic source to the traffic generator
     $traffic attach−agent $source

    #Connect the source and the sink
    $ns connect $source $sink

    return $traffic
}

# Nodes definition
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

# Nodes connection
$ns duplex−link $n0 $n3 1Mb 100ms DropTail
$ns duplex−link $n1 $n3 1Mb 100ms DropTail
$ns duplex−link $n2 $n3 1Mb 100ms DropTail
$ns duplex−link $n3 $n4 1Mb 100ms DropTail

# Nodes position for nam
$ns duplex−link−op $n0 $n3 orient right−down
$ns duplex−link−op $n2 $n3 orient right−up
$ns duplex−link−op $n1 $n3 orient right
$ns duplex−link−op $n3 $n4 orient right

set sink0 [new Agent/LossMonitor]
set sink1 [new Agent/LossMonitor]
set sink2 [new Agent/LossMonitor]
$ns attach−agent $n4 $sink0
$ns attach−agent $n4 $sink1
$ns attach−agent $n4 $sink2

set source0 [attach−expoo−traffic $n0 $sink0 200 2s 1s 100k 1]
set source1 [attach−expoo−traffic $n1 $sink1 200 2s 1s 200k 2]
set source2 [attach−expoo−traffic $n2 $sink2 200 2s 1s 300k 3]

$ns at 0.0 ”record”
```

```
$ns at 10.0 "$source0_start"
$ns at 10.0 "$source1_start"
$ns at 10.0 "$source2_start"
$ns at 50.0 "$source0_stop"
$ns at 50.0 "$source1_stop"
$ns at 50.0 "$source2_stop"
$ns at 60.0 "finish"

$ns run
```