Network Simulator Project
“Earthquake”

Ozan Eren BILGEN
(oe@oebilgen.com)
704031018

ITU Information Institute
Master of Sciences in Computer Sciences
Advanced Topics in Computer Networks Course

Spring 2004

Plan

1. Introduction
2. Proposal
3. Design
4. Observations
    a. Changing Traffic Sources
    b. Changing Queue Type
    c. Changing Record Interval
5. Listing
6. Bibliography

1 Introduction

The aim of this project is to simulate a dynamic network and examine the results of several unattended effects to the flow. ISI Network Simulator (a.k.a. NS) simulates the network, XGraph takes the flow graphics and ISI Network Animator (a.k.a. NAM) presents the visualization.

The environment used to realize the simulation is a RedHat Linux 9.0 running with Linux Kernel 2.6.5 on an Intel Pentium III 550 MHz with 128 MB SDRAM. NS 2.27, NAM 1.10, XGraph 12.1, OTCL 1.8, TCL 8.4.5, TCLCL 1.15, TK 8.4.5, XGraph-12.1, Zlib 1.1.4 are used and all are installed by “ns-all-in-one” package with version number 2.27, which is downloaded from “http://www.isi.edu.tr/ns-nam”.

The main reference to learn NS was Marc Greis’ Tutorial, which is also shipped with NS package. Some of the listing is taken from its tutorial.

2 Proposal

In our network, it happens an earthquake and because of this and its continuing shocks, several links are broken in time. The initial design of city network was as fallowing:

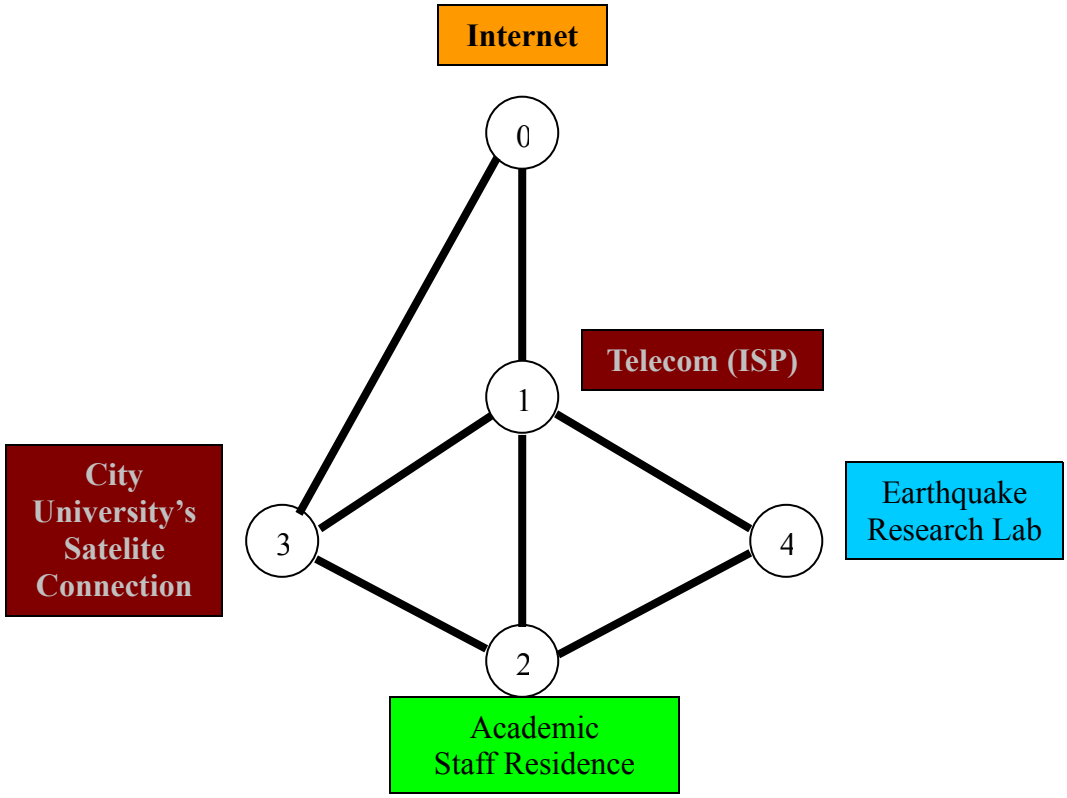

Figure 2.1: Initial network design.

The academic staff, which is in their staff residence, wants to make videoconference over the Internet in order to consulate with other academics in the world about the earthquake and make vital decisions for the city safe. But how they could do this while everything goes down?

All links are the same and have 1MB capacity, 100ms delay and a DropTail queue. Node 2, 3 and 4 are Constant Bit Rate (CBR) UDP sources and are generating an Expoo traffic versus the node 0.

The total simulation time is 5 seconds. When the simulation starts, all three sources also start producing packets. But after 0.5 second, the link 0→3 fails and after 0.5 second from this incident, link 1→2 also stops transmitting. The catastrophe is continuing and after 2 second from the beginning, link 2→3 breaks down. Now is our network as above:

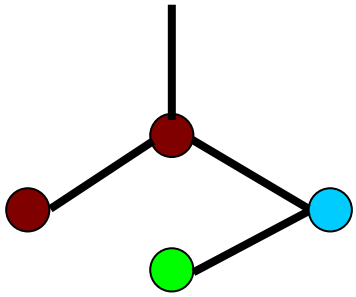

Figure 2.2: Network after 2 seconds.

Then, the engineers finish to repair link 0→3 in 3rd second, which was down in the first 0.5 second. This was a good decision, because link 0→1 breaks down after 0.75 second. Now the network looks like:

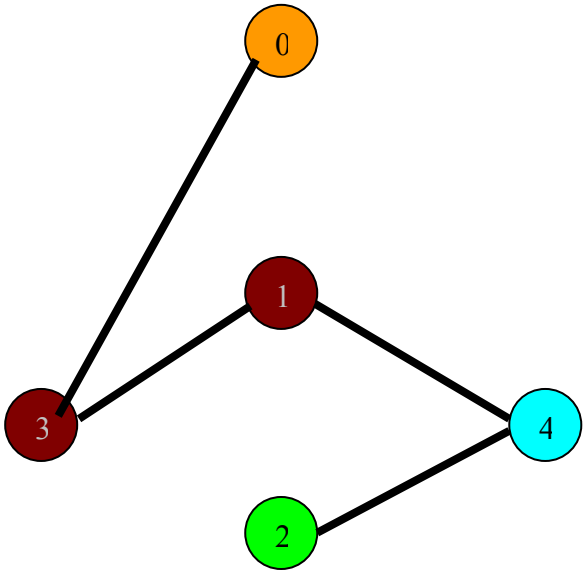

Figure 2.3: Network after 3.75 seconds (final state).

All sources produce packets for 4.5 seconds. This simulation will show us how the three sources are affected by this reorganizations and how many bytes they could send.

3 Design

I designed this network with Anjuta using TCL code. At first, I create a NS object:

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

The sources have to reengineer their paths, because their first decisions may be useless in time. In the tutorial of Mike Greis', it is recommended to add this line in the beginning, just after the creation of NS object.

```
$ns rtproto DV
```

I have three sources and I like to distinguish them while the traffic becomes chaotic. So I assign three colors to them.

```
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green
```

I will use NAM in order to visualize things. I used three files to save their traffic.

```
set nf [open out.nam w]
$ns namtrace-all $nf
set f0 [open out_2-0.tr w]
set f1 [open out_3-0.tr w]
set f2 [open out_4-0.tr w]
```

By the proposal, there are 5 nodes that should be created.

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

The proposal was also fixed the links.

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

While tests, I saw that NAM isn't capable to design how I have imagined the network, so I have to give it some information about the picture.

```
$ns duplex-link-op $n0 $n1 orient down
$ns duplex-link-op $n0 $n3 orient left-down
$ns duplex-link-op $n1 $n2 orient down
$ns duplex-link-op $n1 $n3 orient left
$ns duplex-link-op $n1 $n4 orient right
```

```
$ns duplex-link-op $n2 $n3 orient left-up
$ns duplex-link-op $n2 $n4 orient right-up
```

Then I told NS what to do on exit. I want that it closes the trace files and displays it in a graphical form.

```
proc finish {} {
     global f0 f1 f2
     #Close the output files
     close $f0
     close $f1
     close $f2
     #Call xgraph to display the results
     exec xgraph out_2-0.tr out_3-0.tr out_4-0.tr -geometry 800x400&
  exit 0
}
```

I use “attach-traffic” function to attach traffics to the sources. It takes the source node (2, 3 or 4), the sink, the size of the packages, the burst, the idle time, the rate and the color associated to the traffic.

```
proc attach-traffic { node sink size burst idle rate fid } {
     set ns [Simulator instance]
     set source [new Agent/CBR/UDP]
     $ns attach-agent $node $source
     set traffic [new Traffic/Expoo]
     $traffic set packet-size $size
     $traffic set burst-time $burst
     $traffic set idle-time $idle
     $traffic set rate $rate
     $source set class_ $fid
     $source attach-traffic $traffic
     $ns connect $source $sink
     return $source
}
```

I needed a recording function to see changes.

```
proc record {} {
       global sink2 sink3 sink4 f0 f1 f2
       set ns [Simulator instance]
        set time 0.1
        set bw0 [$sink2 set bytes_]
        set bw1 [$sink3 set bytes_]
        set bw2 [$sink4 set bytes_]
        set now [$ns now]
        puts $f0 "$now [expr $bw0/$time*8/1000000]"
        puts $f1 "$now [expr $bw1/$time*8/1000000]"
        puts $f2 "$now [expr $bw2/$time*8/1000000]"
        $sink2 set bytes_ 0
        $sink3 set bytes_ 0
        $sink4 set bytes_ 0
        $ns at [expr $now+$time] "record"
}
```

I created three traffic sinks and attach them to node 0.

```
set sink2 [new Agent/LossMonitor]
set sink3 [new Agent/LossMonitor]
set sink4 [new Agent/LossMonitor]
   $ns attach-agent $n0 $sink2
   $ns attach-agent $n0 $sink3
   $ns attach-agent $n0 $sink4
```

And created three traffic sources for them.

```
set source0 [attach-traffic $n2 $sink2 10 0.1s 0.05s 150k 1 ]
set source1 [attach-traffic $n3 $sink3 20 0.1s 0.05s 300k 2 ]
set source2 [attach-traffic $n4 $sink4 15 0.1s 0.05s 450k 3 ]
```

I logged everything from the beginning.

```
$ns at 0.0 "record"
```

And finally I was ready to apply the scenario. In the beginning all three starts running.

```
$ns at 0.0 "$source0 start"
$ns at 0.0 "$source1 start"
$ns at 0.0 "$source2 start"
```

But things go wrong and some modifications realize, as explained in the proposal.

```
$ns rtmodel-at 0.5 down $n0 $n3
$ns rtmodel-at 1.0 down $n1 $n2
$ns rtmodel-at 2.0 down $n2 $n3
 $ns rtmodel-at 3.0 up $n0 $n3
$ns rtmodel-at 3.75 down $n0 $n1
```

Then I stop traffic.

```
$ns at 4.5 "$source0 stop"
$ns at 4.5 "$source1 stop"
$ns at 4.5 "$source2 stop"
```

And after 3 seconds, my simulation is finished.

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

At last, I give the command to run the simulation.

```
$ns run
```

4 Observations

I used iterative software development techniques, because it was my first NS simulation. It helped me too much and I didn't lost in the strange NS error messages. As I told in the design section, I learned that I have to give respective node position in order to have something similar to proposals topology, which is a result of this technique. Here, blue, red and green lines are produced by nodes 2, 3 and 4 respectively and XGraph result is:

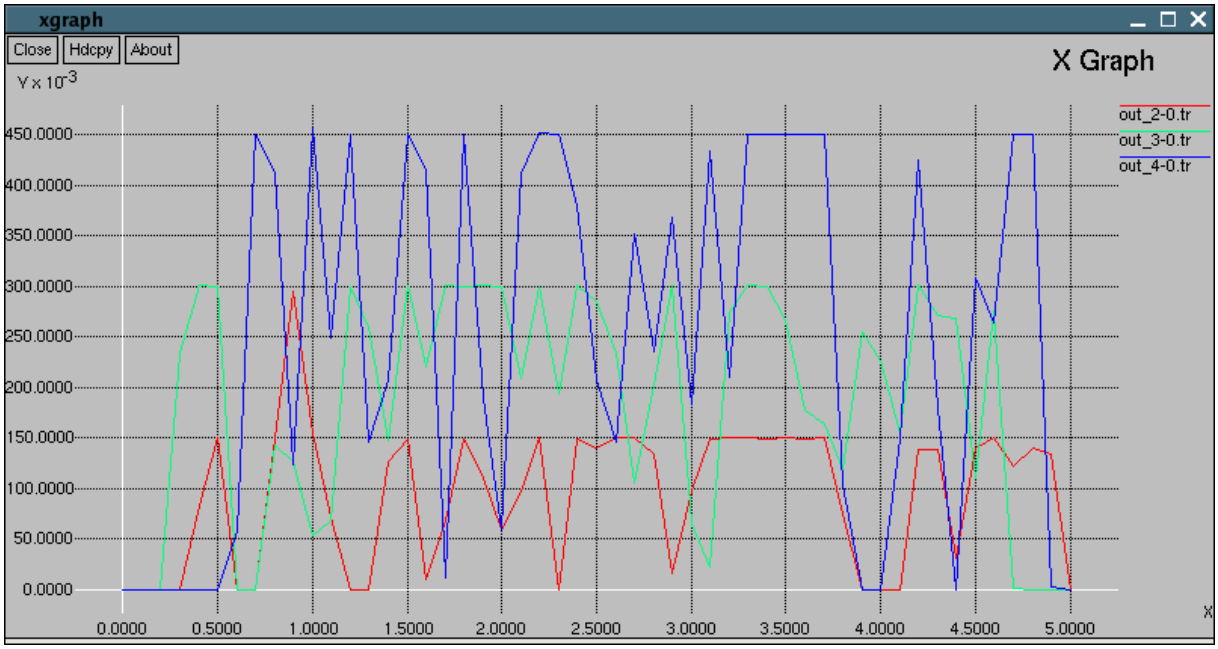

Figure 4.1: XGraph output.

From this graph, I observe that traffic 3→0 has a shortest response time because it has the smallest rate/size ratio. Fallowed by 4→0, the longest response time is calculated for traffic 3 →0, which have an initial path of 4→2→3→0.

The values go to 0 periodically, because they are set to 0 on each call of "record" procedure. If we disable this, we obtain total bytes transmitted by each traffic:

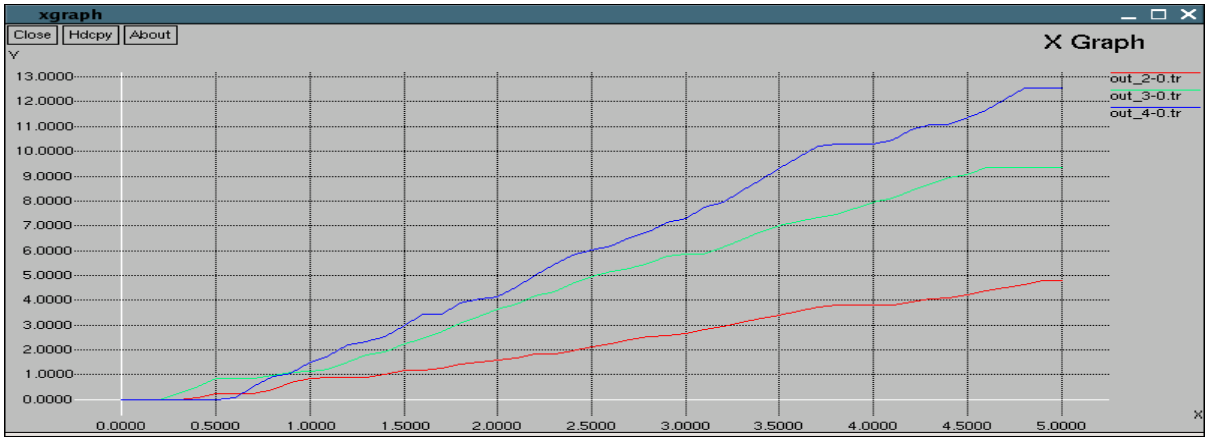

Figure 4.2: XGraph output for total bytes sent.

The blue line has a larger value because its packet size is the smallest. Green and red lines because of the same reason fallow it.

Then I changed some parameters and interpreted the results.

4.a Changing Traffic Sources

I set the fallowing values:

```
set source0 [attach-traffic $n2 $sink2 10 0.1s 0.05s 150k 1 ]
set source1 [attach-traffic $n3 $sink3 10 0.1s 0.05s 150k 2 ]
set source2 [attach-traffic $n4 $sink4 10 0.1s 0.05s 150k 3 ]
```

Now all three produce same traffic. In the result graph, three colors has similar structures:

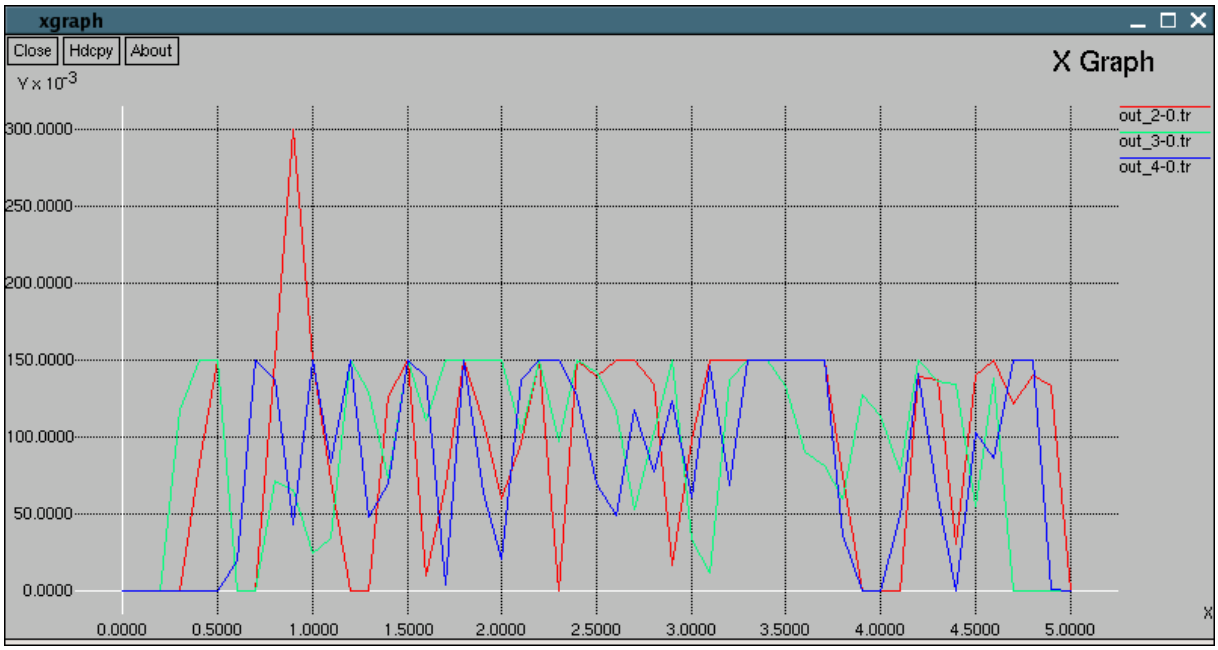

Figure 4.a.1: XGraph output for same traffic.

4.b Changing Queue Type

In order to select Scholastically Fair Queue (SFQ), I hacked fallowing lines:

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

The result of this action is:

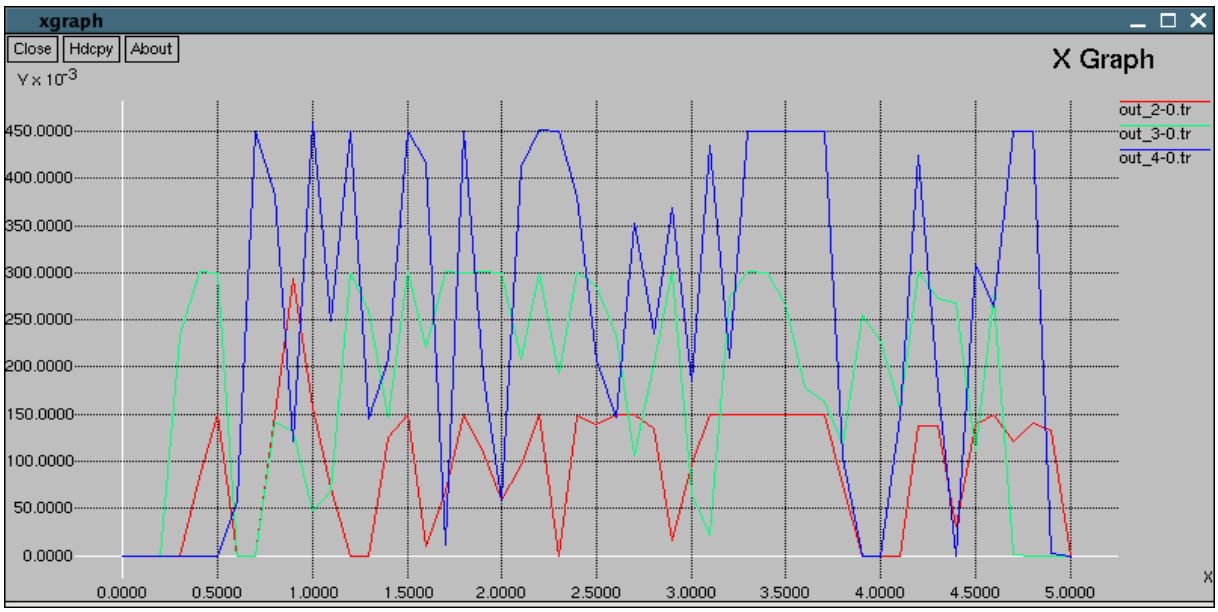

Figure 4.b.1: XGraph output for SFQ.

4.c Changing Record Interval

```
set time 0.5
```

The line above tells NS to record 5 times less then it does for the original code. The result is:

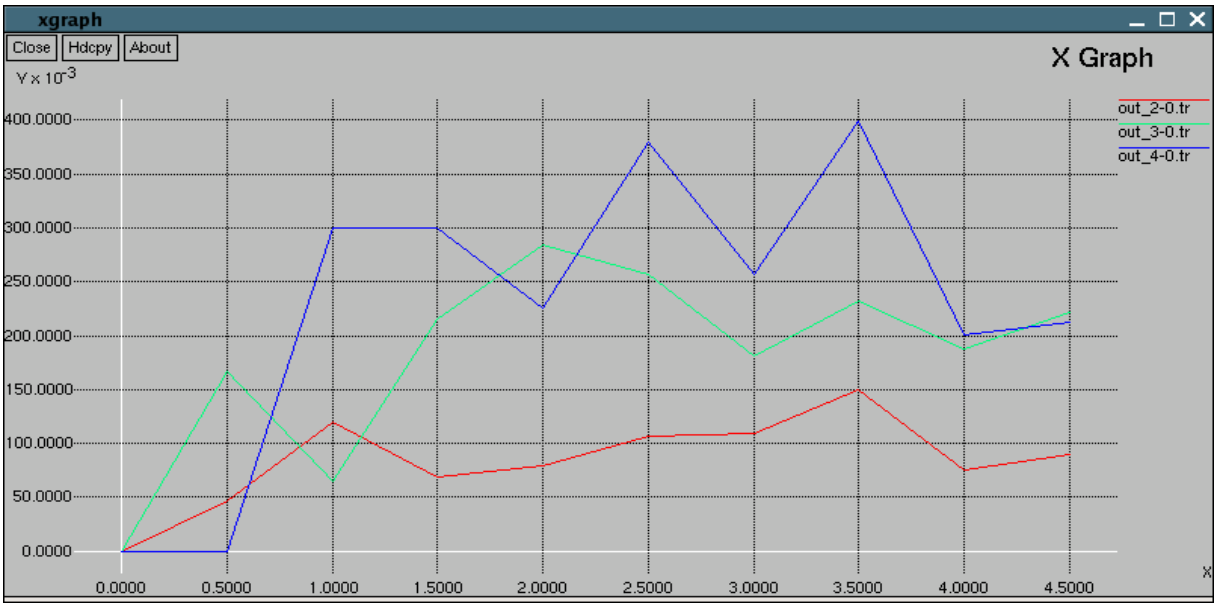

Figure 4.c.1: XGraph output for 0.5 record interval.

This is very obvious because the more we augment the interval, the less we see the small pikes. The consequence comes from record procedure, which resets sent bytes to 0.

5 Listing

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

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

# Prepare NAM output
set nf [open odev.nam w]
$ns namtrace-all $nf

# Open the output files
set f0 [open out_2-0.tr w]
set f1 [open out_3-0.tr w]
set f2 [open out_4-0.tr w]

# Create 5 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]

# Connect the nodes
$ns duplex-link $n0 $n1 1Mb 100ms DropTail
$ns duplex-link $n0 $n3 1Mb 100ms DropTail
$ns duplex-link $n1 $n2 1Mb 100ms DropTail
$ns duplex-link $n1 $n3 1Mb 100ms DropTail
$ns duplex-link $n1 $n4 1Mb 100ms DropTail
$ns duplex-link $n2 $n3 1Mb 100ms DropTail
$ns duplex-link $n2 $n4 1Mb 100ms DropTail

      # Use here for 4.b
      # $ns duplex-link $n0 $n1 1Mb 100ms SFQ
      # $ns duplex-link $n0 $n3 1Mb 100ms SFQ
      # $ns duplex-link $n1 $n2 1Mb 100ms SFQ
      # $ns duplex-link $n1 $n3 1Mb 100ms SFQ
      # $ns duplex-link $n1 $n4 1Mb 100ms SFQ
      # $ns duplex-link $n2 $n3 1Mb 100ms SFQ
      # $ns duplex-link $n2 $n4 1Mb 100ms SFQ

# Organize links
$ns duplex-link-op $n0 $n1 orient down
$ns duplex-link-op $n0 $n3 orient left-down
$ns duplex-link-op $n1 $n2 orient down
$ns duplex-link-op $n1 $n3 orient left
$ns duplex-link-op $n1 $n4 orient right
$ns duplex-link-op $n2 $n3 orient left-up
$ns duplex-link-op $n2 $n4 orient right-up

# Define a 'finish' procedure
proc finish {} {
      global f0 f1 f2
      # Close the output files
      close $f0
      close $f1
```

```
      close $f2
      # Call xgraph to display the results
      exec xgraph out_2-0.tr out_3-0.tr out_4-0.tr -geometry 800x400 &
      exec nam odev.nam &
        exit 0
}

# Define a procedure that attaches a UDP agent to a previously created node
# 'node' and attaches an Expoo traffic generator to the agent with the
# characteristic values 'size' for packet size 'burst' for burst time,
# 'idle' for idle time, 'rate' for burst peak rate and 'fid' for color. The
# procedure connects the source with the previously defined traffic sink
# 'sink' and returns the source object.
proc attach-traffic { node sink size burst idle rate fid } {
      # Get an instance of the simulator
      set ns [Simulator instance]
      # Create a UDP agent and attach it to the node
      set source [new Agent/CBR/UDP]
      # Assign color
      $source set class_ $fid
      # Connect source to node
      $ns attach-agent $node $source
      # Create an Expoo traffic agent and set its configuration parameters
      set traffic [new Traffic/Expoo]
      $traffic set packet-size $size
      $traffic set burst-time $burst
      $traffic set idle-time $idle
      $traffic set rate $rate
      # Attach the traffic agent to the traffic source
      $source attach-traffic $traffic
      # Connect the source and the sink
      $ns connect $source $sink
      return $source
}

# Define a procedure which periodically records the bandwidth received by
# the three traffic sinks sink2/3/4 and writes it to the three files
# f0/1/2.
proc record {} {
      global sink2 sink3 sink4 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 0.1
            # Use here for 4.c
            # set time 0.5
      # How many bytes have been received by the traffic sinks?
      set bw0 [$sink2 set bytes_]
      set bw1 [$sink3 set bytes_]
      set bw2 [$sink4 set bytes_]
      # Get the current time
      set now [$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 --3
      $sink2 set bytes_ 0
      $sink3 set bytes_ 0
      $sink4 set bytes_ 0
      # Re-schedule the procedure
```

```
      $ns at [expr $now+$time] "record"
}

# Create three traffic sinks and attach them to the node n0
set sink2 [new Agent/LossMonitor]
set sink3 [new Agent/LossMonitor]
set sink4 [new Agent/LossMonitor]
$ns attach-agent $n0 $sink2
$ns attach-agent $n0 $sink3
$ns attach-agent $n0 $sink4

# Create three traffic sources
set source0 [attach-traffic $n2 $sink2 10 0.1s 0.05s 150k 1]
set source1 [attach-traffic $n3 $sink3 20 0.1s 0.05s 300k 2]
set source2 [attach-traffic $n4 $sink4 15 0.1s 0.05s 450k 3]

      # Use here for 4.a
      # set source0 [attach-traffic $n2 $sink2 10 0.1s 0.05s 150k 1]
      # set source1 [attach-traffic $n3 $sink3 10 0.1s 0.05s 150k 2]
      # set source2 [attach-traffic $n4 $sink4 10 0.1s 0.05s 150k 3]

# Start logging the received bandwidth
$ns at 0.0 "record"

# Start the traffic sources
$ns at 0.0 "$source0 start"
$ns at 0.0 "$source1 start"
$ns at 0.0 "$source2 start"

# Make link down
$ns rtmodel-at 0.5  down $n0 $n3
$ns rtmodel-at 1.0  down $n1 $n2
$ns rtmodel-at 2.0  down $n2 $n3
$ns rtmodel-at 3.0  up   $n0 $n3
$ns rtmodel-at 3.75 down $n0 $n1

# Stop the traffic sources
$ns at 4.5 "$source0 stop"
$ns at 4.5 "$source1 stop"
$ns at 4.5 "$source2 stop"

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

# Run the simulation
$ns run
```

6 Bibliography

[1] Marc Greis' Tutorial, ISI NS All in One Software, NS Documentation, 2004

[2] Jianping Wang, NS-2 Tutorial, Multimedia Networking Group, The Department of Computer Science, UVA, 2003

[3] NS Fundamentals, Padma Haldar, USC/ISI

[4] NS Fundamentals-2, Padma Haldar, USC/ISI