NS2 Familiarization Report

Group #30

Jiaju Shi

Raghavan S V

Hari Kishan Srikonda

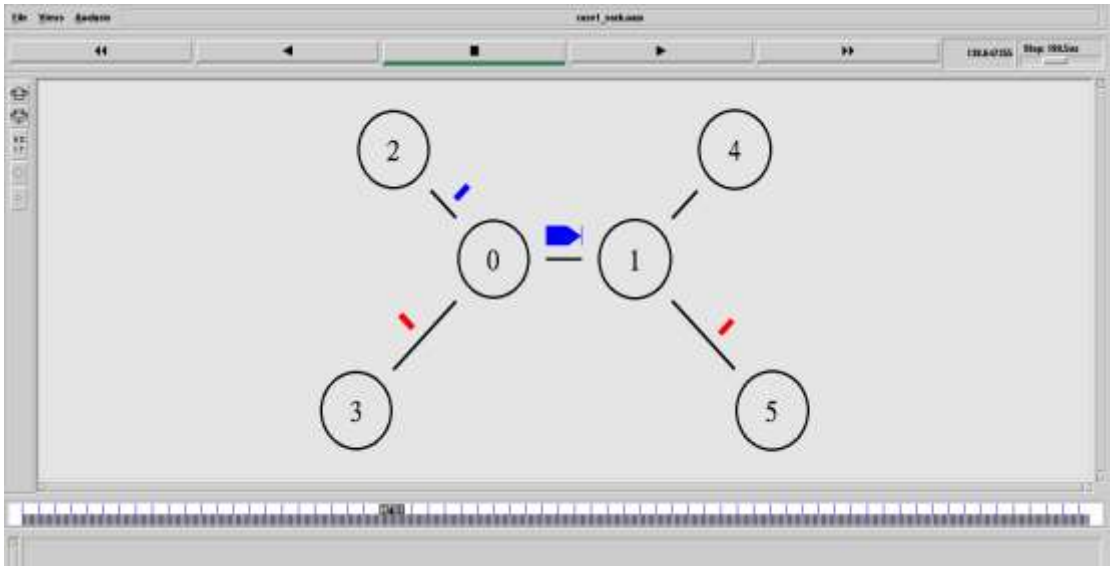
Setup:

Ubuntu 14.04 (64-bit), NS-2, Xgraph, Nam, Tcl/Tk

Procedure:

First, we install ns2 (Network Simulator – 2), NAM (network animator) and Xgraph packages on Ubuntu. Then, by learning the tutorial from the website http://www.isi.edu/nsnam/ns/tutorial/index.html, we start developing the Tcl script for NS-2 simulation. We successfully build the three configurations listed, and run the ns2.tcl script along with building the NAM trace data and output trace file.

The simulation runs for 400 seconds, and we record information of bandwidth from 100 sec to ensure stable values. We further calculate the average throughput for each of the sources (src2 and src3 shown in figure below) and also the ratio of average throughput which are captured in the output as data file.



NAM (NS-2) topology

We consider the TCP flavors: Vegas and Sack1 available in NS-2

- Case 1:
    - src1 - R1 and R2-rcv1 end-2-end delay = 5 ms
    - src2 - R1 and R2-rcv2 end-2-end delay = 12.5 ms
- Case 2:

- src1 - R1 and R2-rcv1 end-2-end delay = 5 ms
- src2 - R1 and R2-rcv2 end-2-end delay = 20 ms

- Case 3:
  - src1 - R1 and R2-rcv1 end-2-end delay = 5 ms
  - src2 - R1 and R2-rcv2 end-2-end delay = 27.5 ms

Results:

TCP flavor: Vegas

	Average Throughput (src1) (MBits/s)	**Average Throughput (src2) (MBits/s)**	**Ratio Avg Throughput (src1/src2)**
Case 1	0.7776540781970056	0.55534148861684474	1.4003169
Case 2	0.91660944635128072	0.41630612312913717	2.2017678
Case 3	0.99988667044374546	0.33302889903668742	3.00240211

It shows the average throughput using TCP Vegas from the recorded data from 100s to 400s. The average throughput ratio of src1 to src2 for RTT_1:2, RTT_1:3, RTT_1:4 are approximately 1.4, 2.2 and 3.0 respectively.

From the above result, when using TCP Vegas, the throughput ratio shows significant difference when RTT increases. It waits for packet loss and only then reduces window size as testament of the TCP Vegas flavor. So the throughput ratio increases, with greater RTT and also, there is no change in window size with increase in RTT.

TCP flavor: Sack1

	Average Throughput (src1) (MBits/s)	**Average Throughput (src2) (MBits/s)**	**Ratio Avg Throughput (src1/src2)**
Case 1	0. 69855271490942306	0. 63443538548716627	1.1010620
Case 2	0. 7277549415018113	0. 60526089130368088	1.2023822
Case 3	0. 75429485683790976	0. 5786932435586517	1.3034451

It shows the average throughput using TCP Sack1 from the recorded data from 100s to 400s. The average throughput ratio of src1 to src2 for RTT_1:2, RTT_1:3, RTT_1:4 are approximately 1.1, 1.2 and 1.3 respectively.

From the above result, when using TCP Sack1, the throughput ratio does not vary much even when RTT increases, so as to ensure fairness which is a primary feature of TCP SACK. Also the throughput ratio only marginally increases, since it waits for packet loss and only then reduces window size. It does not change window size when RTT increases.

In terms of just throughput, TCP Vegas performs better for source 1 whereas TCP SACK performs better for source 2, where the ratio of RTTs are 1:2 for source 1 to source 2. The reason for this is that TCP SACK enhances further on the fairness factor, whereas Vegas is not too considerate about this factor.

Appendix:

We just attach the code of ns2:

```
# Write ns2.tcl code for specified network
# usage:
# ns ns2.tcl <TCP_VERSION> <CASE_NO>
# example: ns ns2.tcl Vegas 2

set ns [new Simulator]

set f0 [open src1_[lindex $argv 0]_[lindex $argv 1].tr w]
set f1 [open src2_[lindex $argv 0]_[lindex $argv 1].tr w]

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

set sum_1 0
set sum_2 0
set count 0
#set throughput_avg [0]

proc finish {} {
        global f0 f1 argv sum_1 sum_2 count
        close $f0
        close $f1
        puts "Average throughput (src1) = [expr $sum_1/$count] MBits/s "
        puts "Average throughput (src2) = [expr $sum_2/$count] MBits/s "
        puts "Ratio of Average throughput (src1/src2) = [expr $sum_1/$sum_2] MBits/s \n"
        #exec nam out.nam &
        #Call xgraph to display the results
       exec xgraph src1_[lindex $argv 0]_[lindex $argv 1].tr src2_[lindex $argv 0]_[lindex $argv 1].tr
-geometry 800x400 &
        exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$ns duplex-link $n0 $n2 10Mb 5ms DropTail
```

```
$ns duplex-link $n3 $n2 1Mb 5ms DropTail
$ns duplex-link $n4 $n3 10Mb 5ms DropTail

if {[lindex $argv 1] == 1} {
$ns duplex-link $n1 $n2 10Mb 12.5ms DropTail
$ns duplex-link $n5 $n3 10Mb 12.5ms DropTail
} elseif {[lindex $argv 1] == 2} {
$ns duplex-link $n1 $n2 10Mb 20.0ms DropTail
$ns duplex-link $n5 $n3 10Mb 20.0ms DropTail
} else {
$ns duplex-link $n1 $n2 10Mb 27.5ms DropTail
$ns duplex-link $n5 $n3 10Mb 27.5ms DropTail
}

#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP/[lindex $argv 0]]
$ns attach-agent $n0 $tcp0

# Create a FTP traffic source and attach it to tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0

#Create a TCP agent and attach it to node n1
set tcp1 [new Agent/TCP/[lindex $argv 0]]
$ns attach-agent $n1 $tcp1

# Create a FTP traffic source and attach it to tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1

set sink0 [new Agent/TCPSink]
set sink1 [new Agent/TCPSink]
$ns attach-agent $n4 $sink0
$ns attach-agent $n5 $sink1

$ns connect $tcp0 $sink0
$ns connect $tcp1 $sink1

proc record {} {
        global sink0 sink1 f0 f1 sum_1 sum_2 count
        #Get an instance of the simulator
        set ns [Simulator instance]
        #Set the time after which the procedure should be called again
        set time 0.01
```

```
        #How many bytes have been received by the traffic sinks?
        set bw0 [$sink0 set bytes_]
        set bw1 [$sink1 set bytes_]
        #Get the current time
        set now [$ns now]
        #Calculate the bandwidth (in MBit/s) and write it to the files
        set sum_1 [expr $sum_1 + $bw0/$time*8/1000000];
        set sum_2 [expr $sum_2 + $bw1/$time*8/1000000];
        set count [expr $count + 1];
      puts $f0 "$now [expr $bw0/$time*8/1000000]"
      puts $f1 "$now [expr $bw1/$time*8/1000000]"
        #Reset the bytes_ values on the traffic sinks
      $sink0 set bytes_ 0
      $sink1 set bytes_ 0
        #Re-schedule the procedure
      $ns at [expr $now+$time] "record"
}

$ns at 100.0 "record"
$ns at 0.0 "$ftp0 start"
$ns at 0.0 "$ftp1 start"
$ns at 400.0 "$ftp0 stop"
$ns at 400.0 "$ftp1 stop"

$tcp0 set class_ 1
$tcp1 set class_ 2
$ns color 1 Blue
$ns color 2 Red

$ns at 400.0 "finish"

$ns run
```