Performance of Competing High Speed TCP Flows with Background Traffic

Master's Project Final Report

Author: Siva Sagar Tella
Email: stella@cs.odu.edu

Project Advisor: Dr. Michele Weigle
Email: mweigle@cs.odu.edu

Project Presentation Date: February 11, 2008

Department of Computer Science
Old Dominion University

TABLE OF CONTENTS

Acknowledgement

I would like to express my thanks to my project advisor, Dr. Michele Weigle, Assistant Professor, Computer Science Department, Old Dominion University for all her support to me in making this project successful. Her guidance to me in completing this project and presenting this project was extremely helpful and commendable.

Abstract

Today, there are new versions of TCP that allow for the transfer of huge amounts of data over high-speed networks. These high-speed TCP protocols use different approaches to utilize the maximum available bandwidth of the network. Since there is not yet an agreed-upon standard, users sharing the same network link may use different high-speed TCP protocols. So two flows using different high-speed protocols may share the same link, competing for bandwidth. Previous work studied the behavior of these high-speed protocols with two flows, each using a different high-speed TCP, but assuming that there is no other traffic on the network, an unlikely situation. We have studied the behavior of these high-speed TCP protocols with the addition of background network traffic. The high-speed TCP protocols included in our study were HS-TCP, Scalable-TCP, CUBIC and FAST. We used network simulator ns-2 to simulate the network and high-speed TCP flows. The Tmix tool in ns-2 is used to generate the background traffic.

1. Introduction

Standard TCP is the most widely used protocol to handle most Internet traffic. Standard TCP has some limitations when used on high bandwidth delay product links. It has a problem with the utilization of the large amounts of available bandwidth. A number of high speed versions have been proposed to the Standard TCP to address this problem. Some of the important high speed TCP versions are HS-TCP, Scalable TCP, CUBIC and FAST. These high speed versions of TCP use different approaches to improve the utilization of available bandwidth. In this project, these high speed TCP protocols are tested against each other by studying the behavior of the competing flows using these high speed protocols.

In previous work [1, 2], a set of experiments were run in the NS-2 simulator where two competing high speed TCP flows were simulated and their behavior was traced for the study. The two high speed flows were simulated in such a way that they were started at different times. It was assumed that there is no traffic other than the two high speed flows in the network.

Motivation: Assuming that there is no traffic, other than the two high speed flows in the network is not a realistic scenario. To be more realistic, we introduced background

traffic along with the high speed TCP flows in the network. Our aim is to study if background traffic has any effect on the behavior of the competing high speed TCP flows and how the fairness in sharing the available bandwidth by two high speed flows is affected. We have studied the behavior of four high speed protocols HS-TCP, CUBIC, Scalable – TCP and FAST with the background network traffic.

In previous work [1, 2], all possible combinations of HS-TCP, Scalable-TCP, FAST, H-TCP, BIC-TCP and CUBIC were run in ns-2 network simulator using the topology shown in Figure 1.

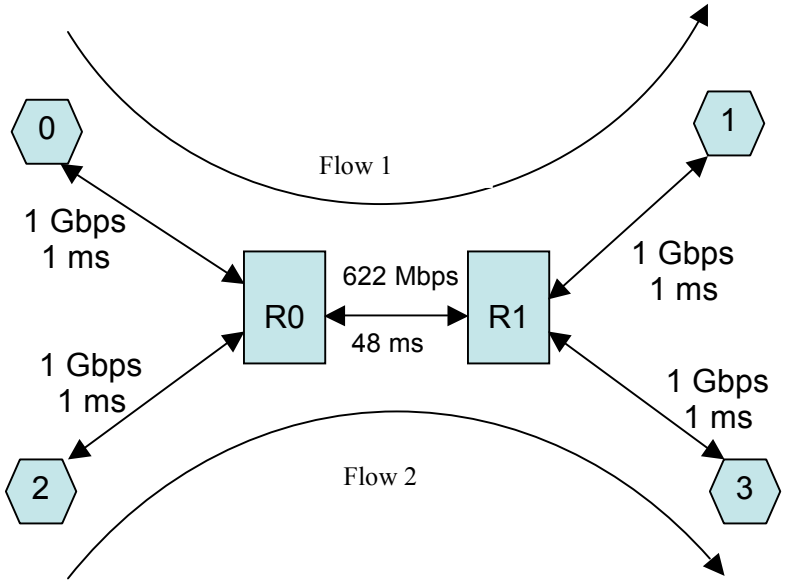

Figure 1: Network Topology as in [1]

The simulation was run for 500 seconds with Flow 1 starting at 0 and Flow 2 starting at 50 seconds. It was found that inter-protocol (flows of two different high-speed versions) behavior was not fair. When two flows are started at different times, the fairness of intra-protocol (flows of the same high-speed version) behavior is also suffered due to slow convergence times. Scalable-TCP and FAST do not consider the other competing flow. Scalable-TCP is very aggressive in obtaining the bandwidth even when competing with another Scalable-TCP flow.

2. Background

A number of high-speed versions of standard TCP have been put forward of which High-Speed TCP (HS-TCP), Scalable-TCP (S-TCP), CUBIC and FAST are prominent. All these high speed versions obtain bandwidth faster than standard TCP.

High-speed protocols can be divided in to loss-based, that is those depend upon packet drops for detecting congestion and delay-based, those that depend on queuing delays in addition to packet drops to detect congestion. HS-TCP, S-TCP and CUBIC are loss-based and FAST is delay-based. Based on the type of increase and decrease parameters they use to increase or decrease the congestion window, high speed TCP protocols are categorized in to static approach protocols and dynamic approach protocols. Static approach use fixed increase and decrease parameters. Dynamic approach use parameters that depends upon some other variables like previous congestion window, time since last packet drop etc.

2.1 Overview of High Speed TCPs

2.1.1 High-Speed TCP (HS-TCP)

HS-TCP [7] behaves more aggressively than standard TCP in low loss rate environments and the same as standard TCP in high loss rate environments. It uses standard TCP's congestion control algorithm below a threshold value which is LowWindow. HS-TCP increases its congestion window when an acknowledgement is received and decreases its congestion window when a packet loss is detected.

Congestion window is varied as following:

$$cwnd = cwnd + \alpha / cwnd \quad \text{(when an acknowledgement is received)}$$

$$cwnd = cwnd * (1 - \beta) \quad \text{(when a packet drop is detected)}$$

In the above equations α and β are called increase and decrease parameters which are maintained in a lookup table.

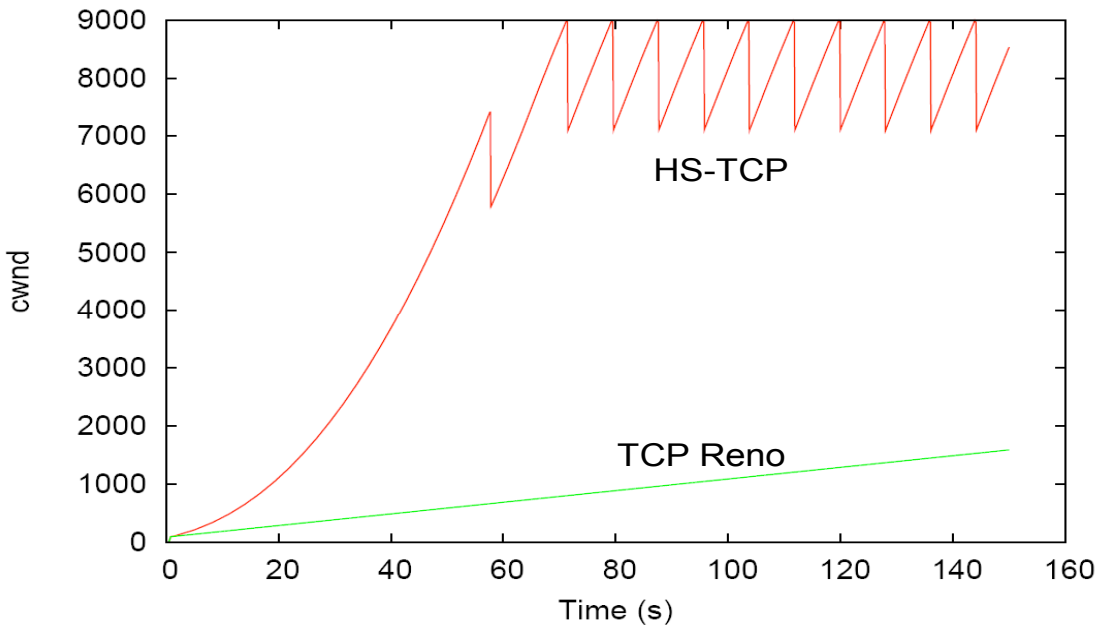

Figure 2: cwnd of HS-TCP vs TCP Reno

In Figure 2, congestion window size of HS-TCP and TCP Reno is plotted. The HS-TCP flow and TCP-Reno flow are run separately. In this experiment, a packet drop is forced initially (not seen clearly in the graph). You can observe how fast HS-TCP gained the available bandwidth after a packet drop. Where as TCP Reno flow could not utilize the bandwidth.

2.1.2 Scalable – TCP

Scalable-TCP [5] was proposed as an improvement to HS-TCP and behaves in the same way as HS-TCP with an exception in determining the values of increase and decrease parameters of the congestion window. It uses 0.01 and 0.125 as fixed increase and decrease parameters respectively which leads to more aggressiveness in increasing and less aggressiveness in decreasing the congestion window than Standard TCP.

Congestion window is varied as following:

cwnd = cwnd + 0.01 (when an acknowledgement is received)

cwnd = (1 – 0.125) cwnd (when a packet drop is detected)

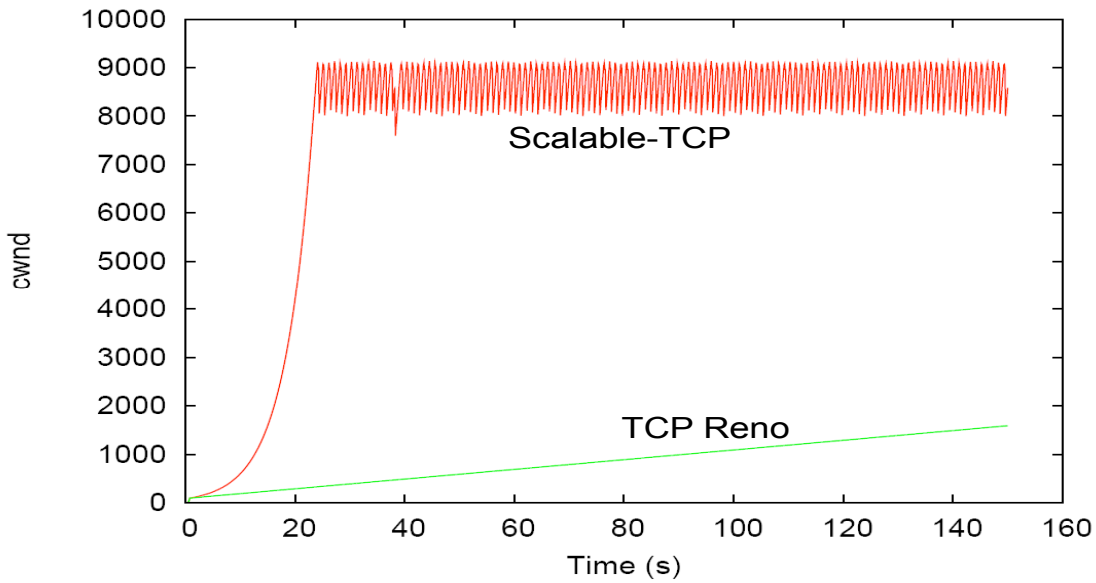

Figure 3: cwnd of Scalable-TCP vs TCP Reno

In Figure 3, congestion window size of Scalable-TCP and TCP Reno is plotted. The Scalable-TCP flow and TCP-Reno flow are run separately. In this experiment, a packet drop is forced initially (not seen clearly in the graph). You can observe how instantaneously Scalable-TCP has gained the available bandwidth due to its fixed nature in increasing the congestion window after a packet drop. Where as TCP Reno flow could not occupy the bandwidth to a fair extent.

2.1.3 CUBIC

CUBIC [6] high speed TCP, as its name suggests, uses a cubic function to increase the congestion window when an acknowledgement is received. CUBIC varies its congestion window based on the time elapsed since last packet drop has occurred. This makes congestion window to grow faster after a packet drop. And also the congestion window rate slows down as it gets closer to the value before the last packet drop occurred. The dependency of CUBIC on the time elapsed since last packet drop makes it

fair regardless of its round-trip time (RTT). Because, no matter what the RTT of the two flows may be, the time elapsed from last packet drop will be same for both flows. There is a limit on the maximum size of the congestion window.

The congestion window of CUBIC (W_{cubic}) is determined by the following equation:

$$W_{cubic} = C\,(t - K)^3 + W_{max}$$

C is a scaling factor

t is the time elapsed since last packet drop

W_{max} is the congestion window size just before last reduction

K is a factor determined as

$$K = (W_{max}\,\beta / C)^{1/3}$$

β is the constant decrease factor for window reduction during a packet drop.

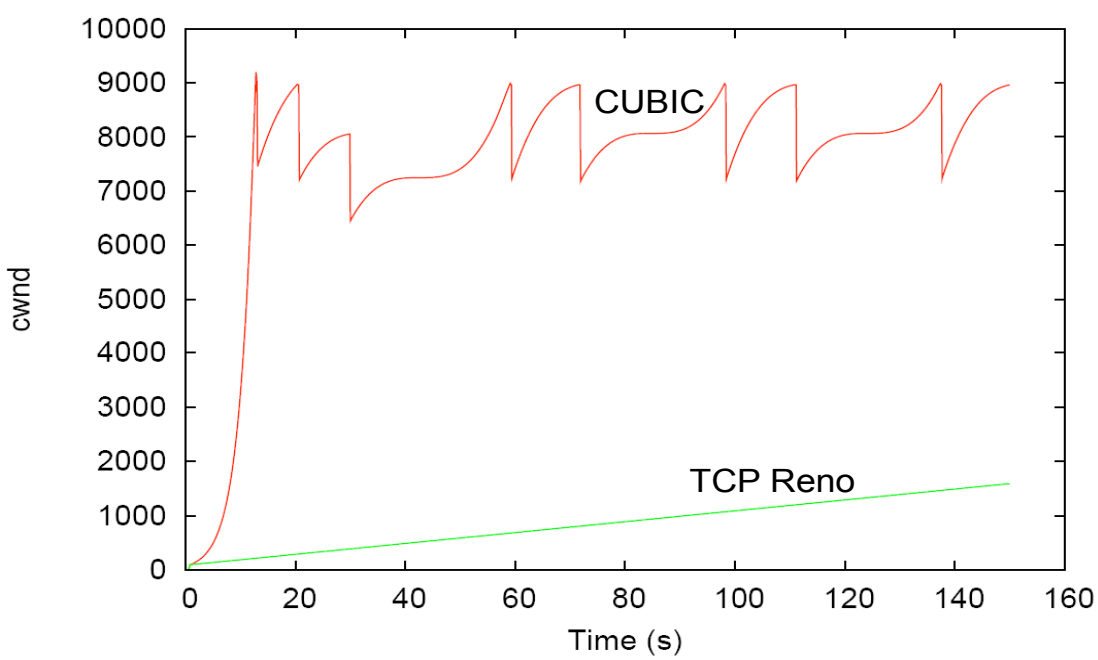

Figure 4: cwnd of CUBIC vs TCP Reno

In Figure 4, congestion window size of CUBIC and TCP Reno is plotted against the time (in seconds). The CUBIC flow and TCP-Reno flow are run separately. In this experiment, a packet drop is forced initially (not seen clearly in the graph). You can observe how fast CUBIC flow gained the available bandwidth after a packet drop, where as TCP Reno flow could occupy only a small portion of the available bandwidth in the given time.

2.1.4. FAST TCP

FAST [4] is a delay-based protocol. The level of congestion in the network is determined by observing the RTT of the flow. FAST can detect the congestion in the network earlier than other protocols as delay can be sensed faster than loss in the network. FAST updates the congestion window depending on the average RTT and a factor which determines the number of packets it maintains in the router queues (α). FAST operates in a more aggressive mode as long as the RTT remains at a minimum value. As congestion builds up and the router queues are filled, leading to higher RTTs, FAST switches to less aggressive mode. FAST does not react to transient congestion as it varies its congestion window based on the average RTT.

The congestion window of FAST is determined as following:

$$W = \min \{2w, (1-\gamma)\, w + \gamma\, ((\text{baseRTT} / \text{RTT})\, w + \alpha)\}$$

γ is a constant whose value is (0,1]

baseRTT is the minimum RTT observed

RTT is the current average round trip time

α is the number of packets maintained in a queue

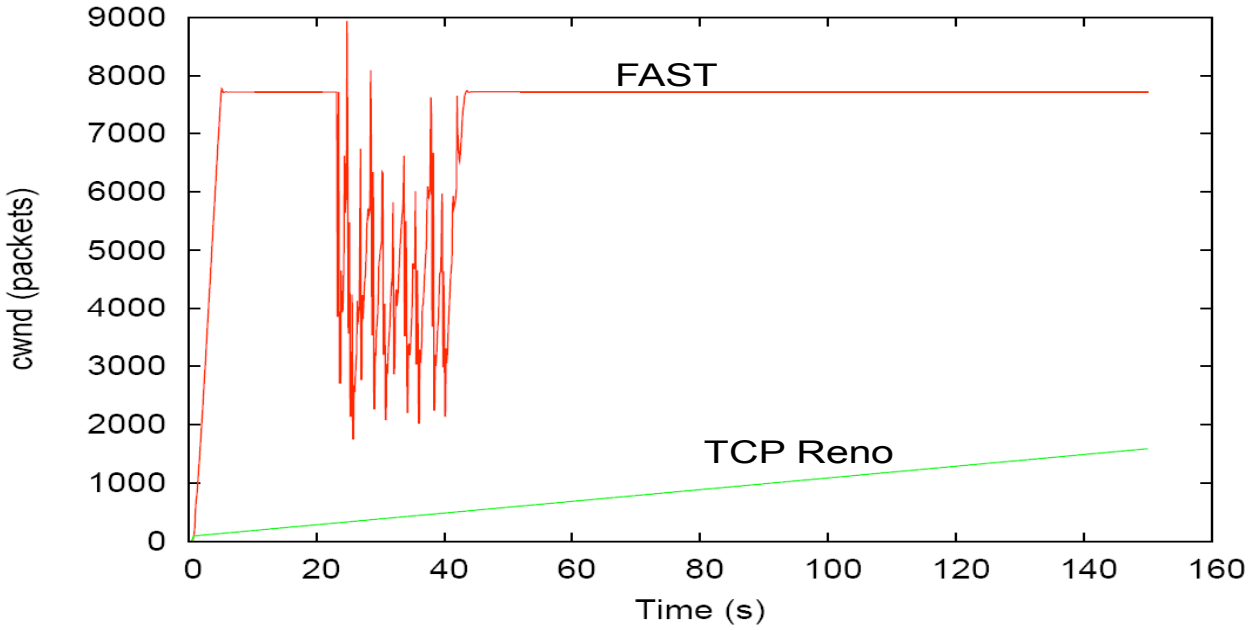

Figure 5: cwnd of FAST vs TCP Reno

In Figure 5, congestion window size of FAST and TCP Reno is plotted. The FAST flow and TCP-Reno flow are run separately. In this experiment, a packet drop is forced initially (not seen clearly in the graph). Here FAST does not react to the initial packet drop as it does not react to the transient congestion. You can observe how FAST gained the available bandwidth instantaneously due to this. Where as TCP Reno flow could occupy only a small amount of bandwidth for the given time.

2.2 Background Traffic – Tmix Traffic Generation

Tmix [10] is a traffic generation model that can produce realistic Internet traffic, which is heterogeneous in nature. It provides traffic as generated by a mix of various TCP applications such as web, email and P2P file sharing.

2.2.1 Tmix Module in NS-2

The Tmix module [3, 10] in NS-2 is used to generate network traffic which is statistically similar to that observed on a network link. The characteristics of the traffic on a network are represented as connection vectors. These connections vectors are provided as input to the Tmix module of NS-2.

The Tmix module in NS-2 has two primary components. They are TmixNode and Tmix_DelayBox. The TmixNode component is responsible for generating the traffic flow. Each TmixNode has two instances, initiator and acceptor. The initiator is the one which starts the connection. Data packets and ACKs flow between initiator and acceptor in both directions. The Tmix_DelayBox component is responsible for introducing the delay factor to the flows.

3. Methodology

3.1 Experiment Setup

The objective of the experiment is to study the behavior of the high speed TCP flows when they compete against each other with background network traffic that is generated using Tmix. A very simple model which has two high speed flows traveling in the same direction with background traffic in both directions is studied. The minimum RTT of the flows is fixed based on the propagation delay of the links, and the starting times of the flows are also fixed.

All experiments have been run in the NS-2 network simulator version 2.27. All the protocols that we have studied are implemented in NS-2. NS-2 implementations of Scalable-TCP, CUBIC from [8], FAST from [9] and HS-TCP have been used in the study.

3.2 Network Simulation Setup

Figure 6 shows the network simulation setup. The network topology uses two 'TmixNode' instances to create bi-directional traffic. Four nodes 0, 1, 2 and 3 are created. Node 0 and Node 3 are set as initiators and Node 1 and Node 2 are set as acceptors. Nodes 0 and 1 are controlled by one instance of TmixNode and nodes 2 and 3 are controlled by second instance of TmixNode. Node R0 (4) is a Tmix_DelayBox, that delays the packets in the flows between nodes 0 and 1, similarly node R1 (5) (Tmix_DelayBox) delays the packets between the nodes 3 and 2. The links between 0 and R0, 2 and R0, 1 and R1 and 3 and R1 have capacity of 1 Gbps and a propagation delay of 1 millisecond (ms). The link between R0 and R1 has a capacity of 622Mbps and a propagation delay of 48 ms. This results in a minimum RTT of 100 ms for each flow.

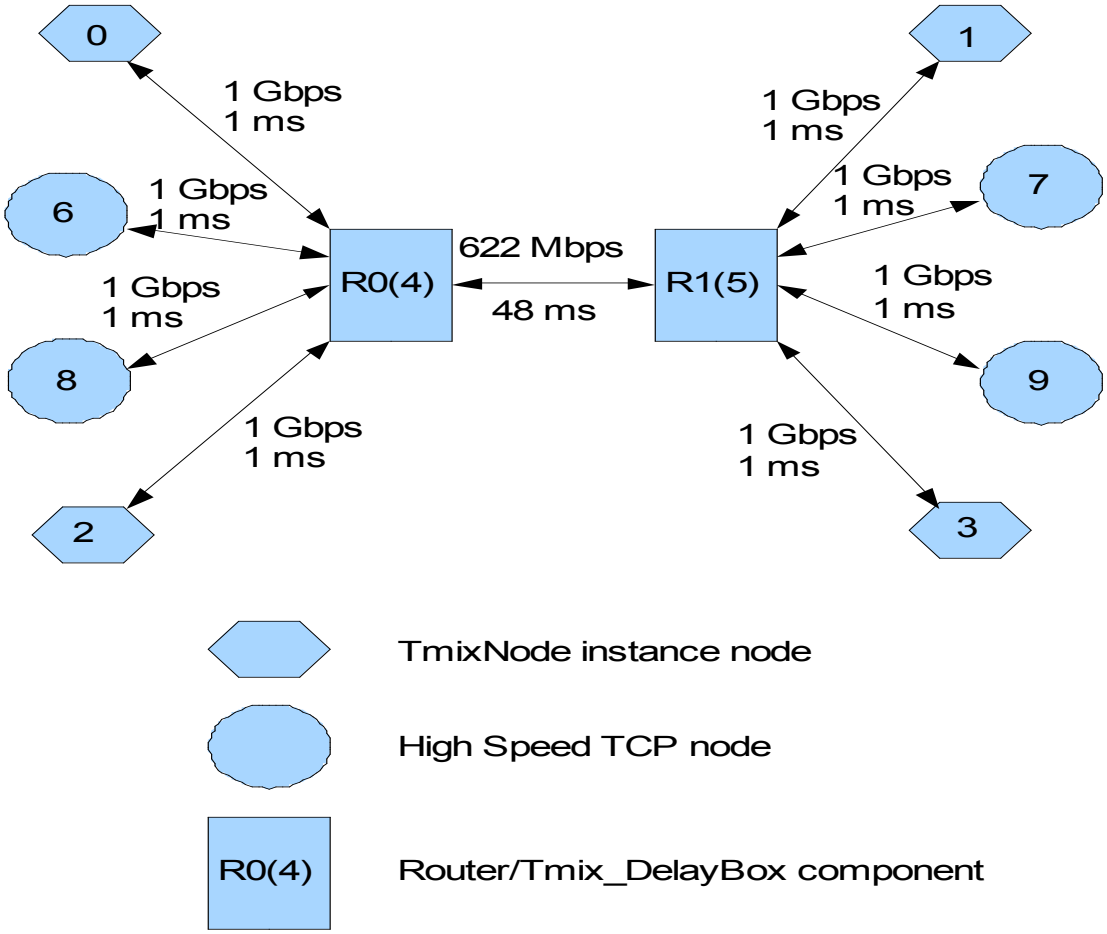

Figure 6: Network Simulation Setup

Now the high speed flows are introduced into the above network setup. The topology of the high speed flows is as follows: nodes 6 and 8 are the sources and nodes 7 and 9 are the sinks. Nodes 6 and 8 are connected to the Tmix_DelayBox, R0 (4) by a 1 Gbps link with 1 ms propagation delay and Nodes 7 and 9 are connected to the Tmix_DelayBox, R1 (5) by a 1 Gbps link with 1 ms propagation delay.

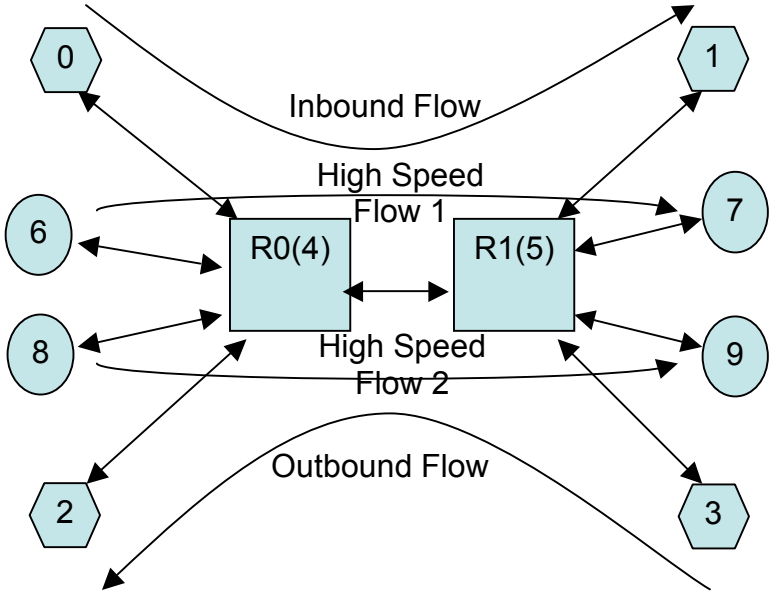

Figure 7: Traffic Flows in the network

Figure 7 shows the traffic flows in the network setup. Background traffic has two flows, inbound flow which flows from node 0 to node 1 and outbound flow which flows from node 3 to node 2. High speed flow 1 flows from node 6 to node 7 and high speed flow 2 flows from node 8 to node 9.

3.3 Experiments Run

We tested the CUBIC, Scalable-TCP, HS-TCP and FAST high-speed TCP protocols against each other to see how fair they are to each other with background network traffic. All the 16 possible combinations with the above four protocols are tested. Two high-speed TCP flows, each using one of the above four protocols are run across the single link between R0 and R1. First the Tmix traffic (background traffic) is started at time 0. The Tmix traffic is allowed to run for 400 seconds to remove transient startup effects. At time 400 seconds the first high speed TCP flow is started. The second flow is started 50 seconds after the first flow start time (i.e. at 450 seconds) so that first flow is allowed to occupy its share of the bandwidth before the second flow starts.

3.4 Asymmetry Metric

The previous study [1] used the asymmetry metric, A, as a metric of fairness. It is calculated as the ratio of the difference of average throughput of flow 1 and flow 2 (t_1- t_2) to sum of the average throughput of two flows (t_1 + t_2).

$$A = \frac{t_1 - t_2}{t_1 + t_2}$$

t_1 throughput of Flow 1

t_2 throughput of Flow 2

If A value is greater than 0 (A > 0) then flow 1 dominates flow 2 in occupying the bandwidth. If A value is less than 0 (A < 0) then flow 2 dominates flow 1. The closer A value is to 0, the more fairly bandwidth is shared. The closer A value is to 1, the more unfairly bandwidth is shared.

3.5 Throughput of Background Traffic

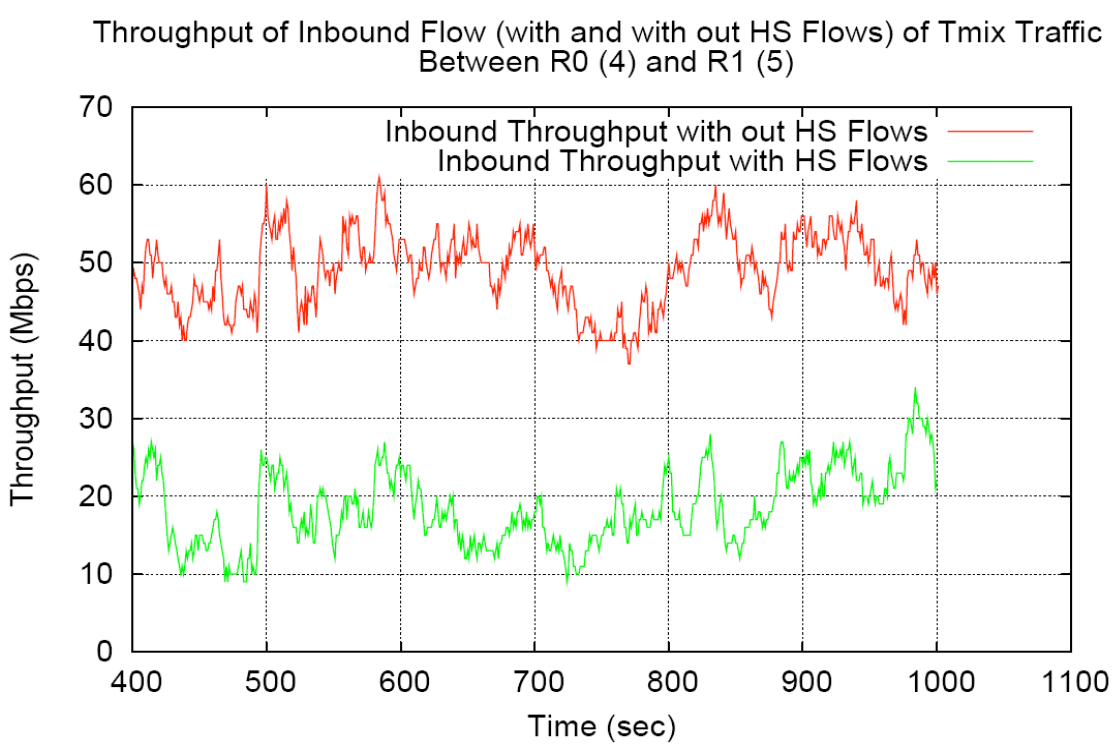

Figure 8: Throughput of Inbound traffic with and without High Speed Flows

In Figure 8, the red color (darker) line is the inbound flow throughput of Tmix traffic when there are no high speed flows in the network. The green color (lighter) line is the inbound flow throughput with high speed flows introduced. So the high speed flows have grabbed some bandwidth from the background traffic. The graph is plotted from 400 seconds as the first high speed flow is introduced at time 400.

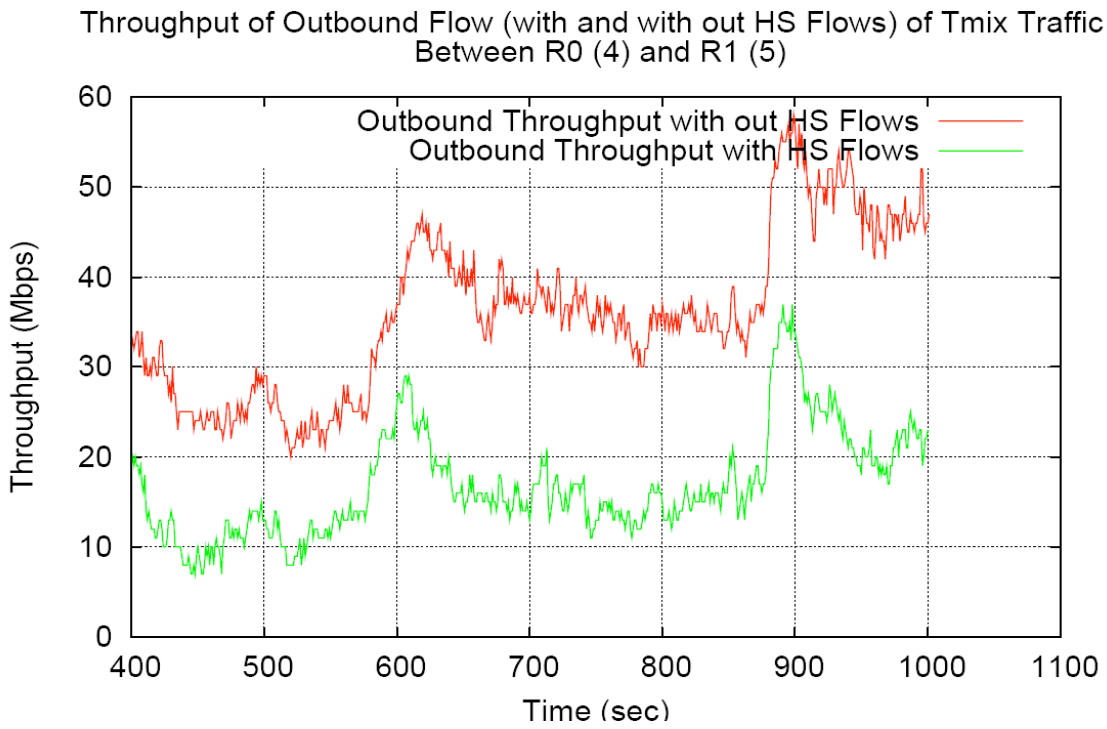

Figure 9: Throughput of Outbound traffic with and without High Speed Flows

In Figure 9, the red color (darker) line is the outbound flow throughput of Tmix traffic when there are no high speed flows in the network. The green color (lighter) line is the outbound flow throughput with high speed flows introduced. So as with inbound traffic, the high speed flows have grabbed some bandwidth of the background traffic.

4. Results of Experiments

Table 1, shows the asymmetry values of all 16 combinations of experiments. Column 2 has values calculated without background traffic, as reported in the previous study. The cases that will be discussed in the following sections are highlighted.

Protocols in Simulation Flow1 – Flow2	Asymmetric Value without Background Traffic	Asymmetric Value with Background Traffic
Scalable-HS	0.996	0.762
Scalable-FAST	0.488	0.932
Scalable-CUBIC	0.987	0.857
Scalable-Scalable	0.975	-0.023
HS-Scalable	-0.996	-0.803
HS-FAST	0.451	0.891
HS-CUBIC	0.599	0.270
HS-HS	0.074	-0.123
FAST-Scalable	-0.489	-0.921
FAST-HS	-0.452	-0.891
FAST-CUBIC	-0.487	-0.890
FAST-FAST	-0.233	0.039
CUBIC-Scalable	-0.978	-0.877
CUBIC-HS	-0.481	-0.358
CUBIC-FAST	0.489	0.889
CUBIC-CUBIC	0.08	-0.031

Table 1: Asymmetry values of all the experiments

4.1 Scalable – HS-TCP Simulation

Figure 11 and Figure 12 show the graphs of Scalable – HS-TCP simulation with and without background traffic, respectively. In these figures the red (darker) line represents the Scalable flow (flow 1) and green (lighter) line represents the HS-TCP flow (flow 2).

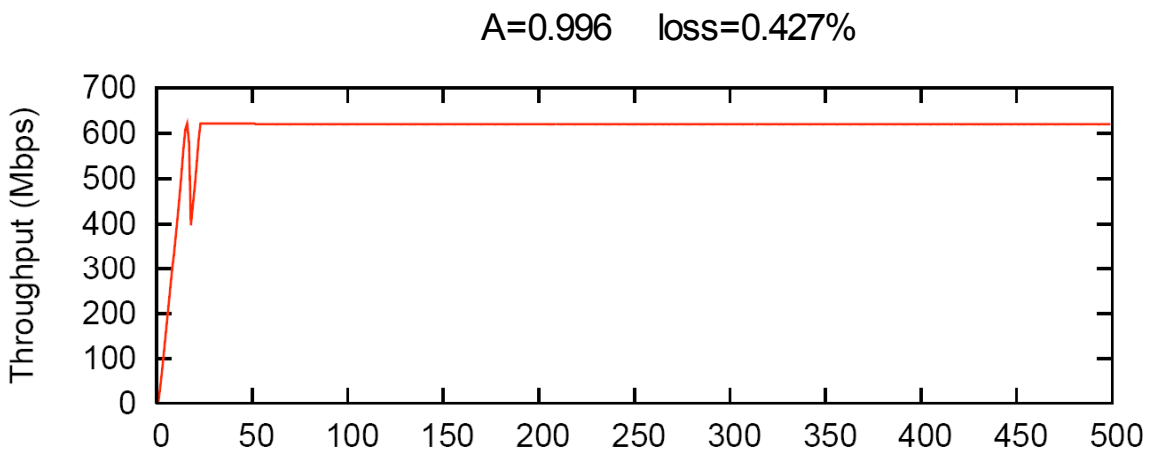

Figure 11: Scalable – HS-TCP Simulation without Background Traffic

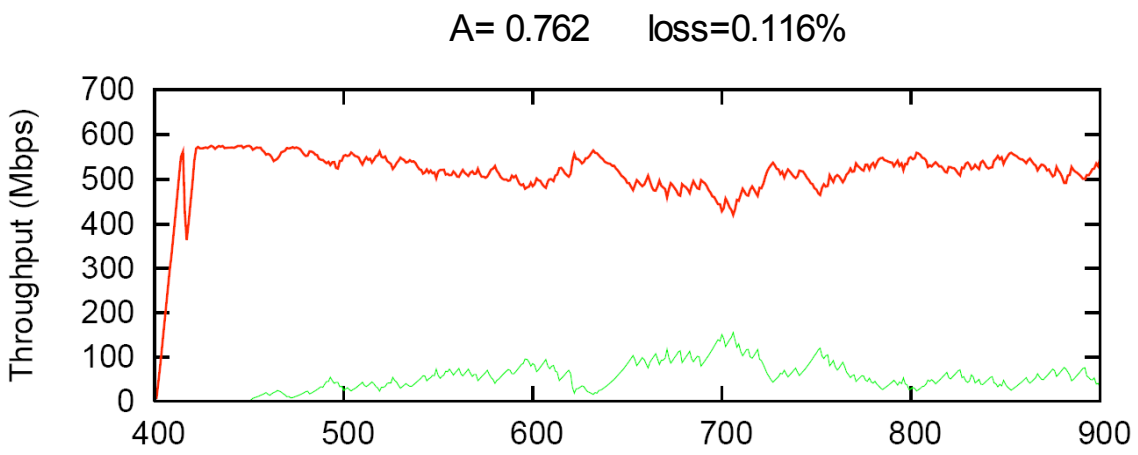

Figure 12: Scalable – HS-TCP Simulation with Background Traffic

In Figure 11, there is no flow 2 (green (lighter) lines visible. So without background traffic, the Scalable flow has totally suppressed HS-TCP flow (flow 2). When background traffic is added (Figure 12), the HS-TCP flow (flow 2) has some share of the bandwidth. So with background traffic added, the fairness in sharing the bandwidth in this case is improved. This may be because, with the addition of background traffic, the round trip time (RTT) increases as the queuing delay is increased. With the increase in RTT, Scalable TCP (flow 1) becomes less aggressive. As a result HS-TCP (flow 2) has a chance to acquire some amount of bandwidth.

4.2 Scalable – Scalable Simulation

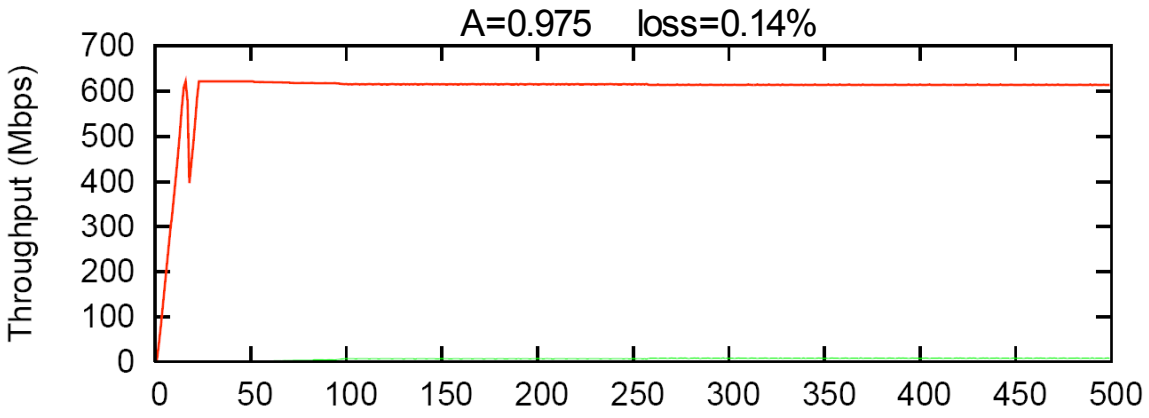

Figure 13: Scalable - Scalable Simulation without Background Traffic

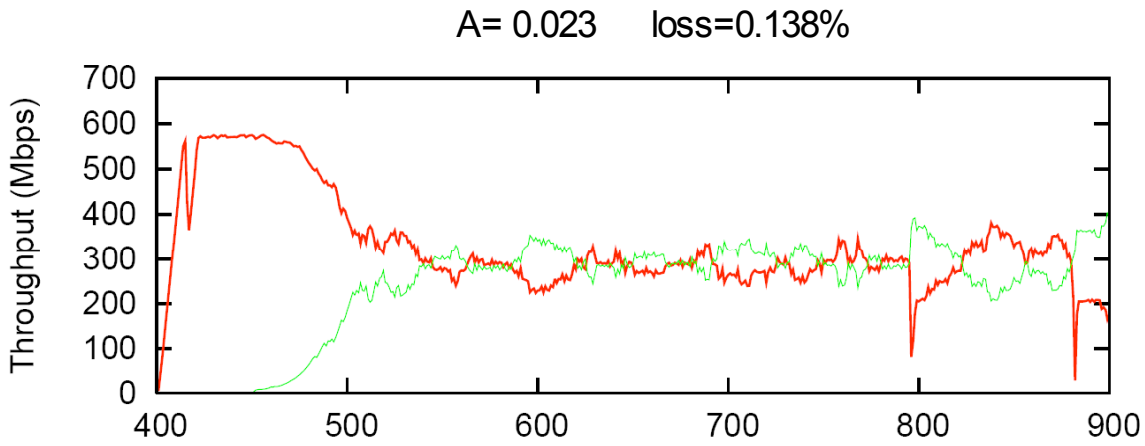

Figure 14: Scalable - Scalable Simulation with Background Traffic

Figure 13 and Figure 14 show the graphs of the Scalable – Scalable simulation with and without background traffic respectively. In these figures the red (darker) line represents Scalable flow (flow 1) and green (lighter) line represents another Scalable flow (flow 2).

In Figure 13, flow 2 was almost suppressed by flow 1 though two flows are of the same protocol. But when background traffic is added, the two flows almost share the bandwidth equally as shown in Figure 14. This may be due to increase in the round trip

time (RTT) as queuing delay is increased because of the addition of background traffic. As a result flow 1 loses the dominance over flow 2.

4.3 HS-TCP – CUBIC Simulation

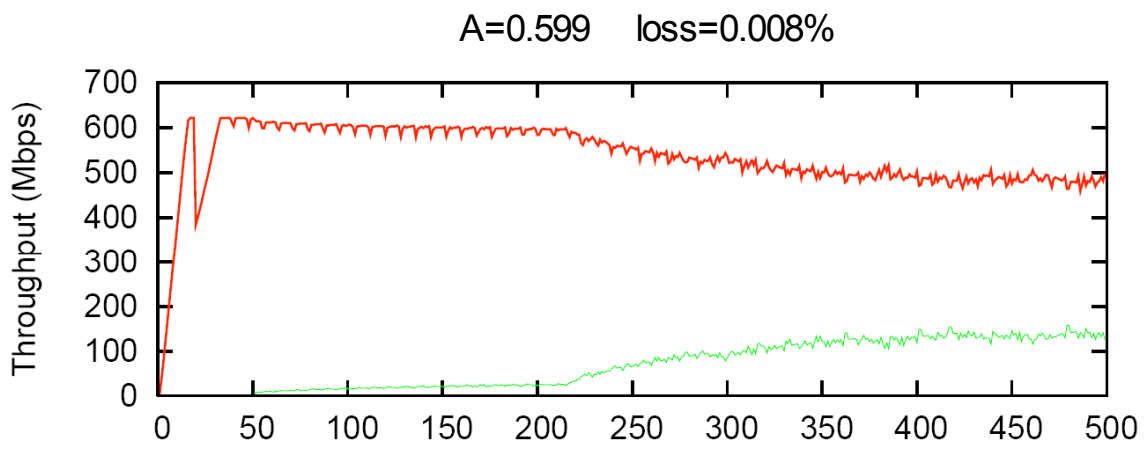

Figure 15: HS-TCP – CUBIC Simulation without Background Traffic

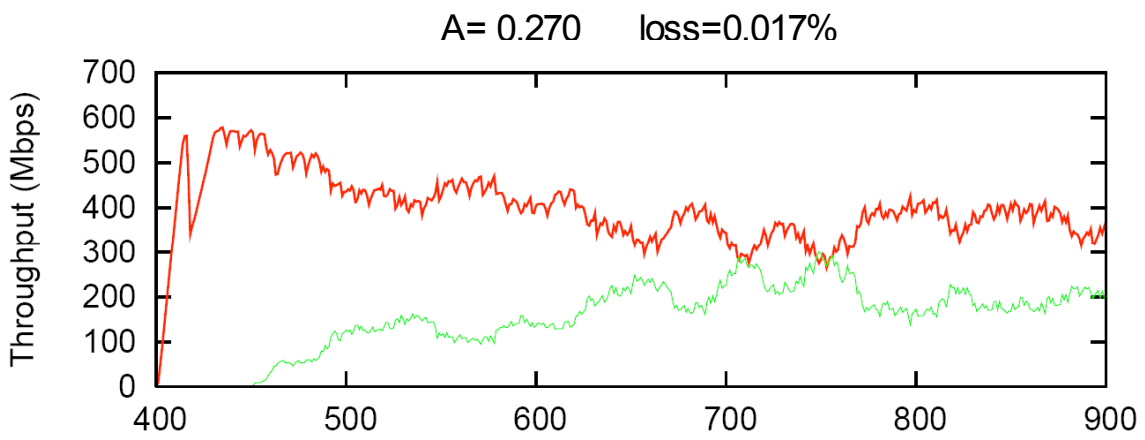

Figure 16: HS-TCP – CUBIC Simulation with Background Traffic

Figure 15 and Figure 16 show the graphs of the HS-TCP – CUBIC simulation with and without background traffic respectively. In these figures the red (darker) line represents HS-TCP flow (flow 1) and green (lighter) line represents CUBIC flow (flow2).

When Figure 15 and Figure 16 are compared, we can see a fair share of the bandwidth in the case of with background traffic (Figure 16). This may be due to the increase in packet loss rate at router buffers with added background traffic to the network. HS-TCP behaves less aggressively in higher loss rate environments.

4.4 CUBIC – FAST Simulation

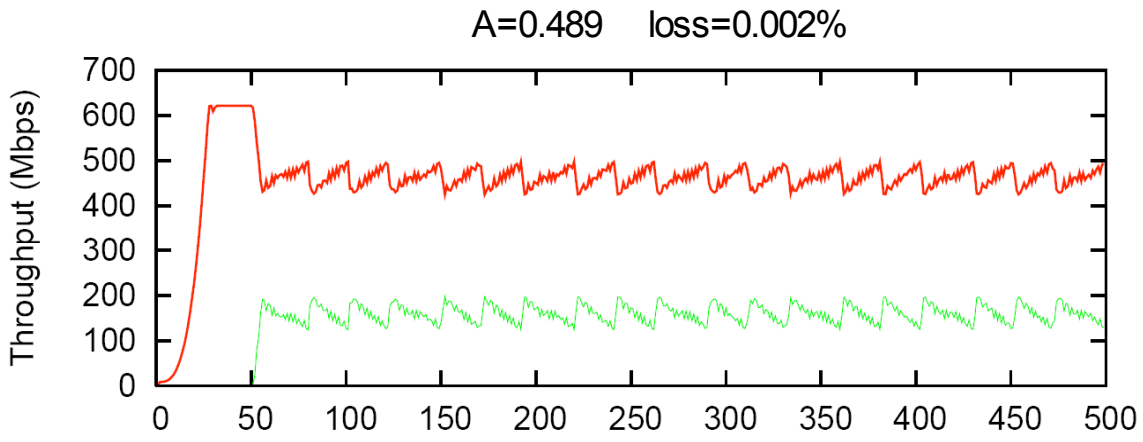

Figure 17: CUBIC - FAST Simulation without Background Traffic

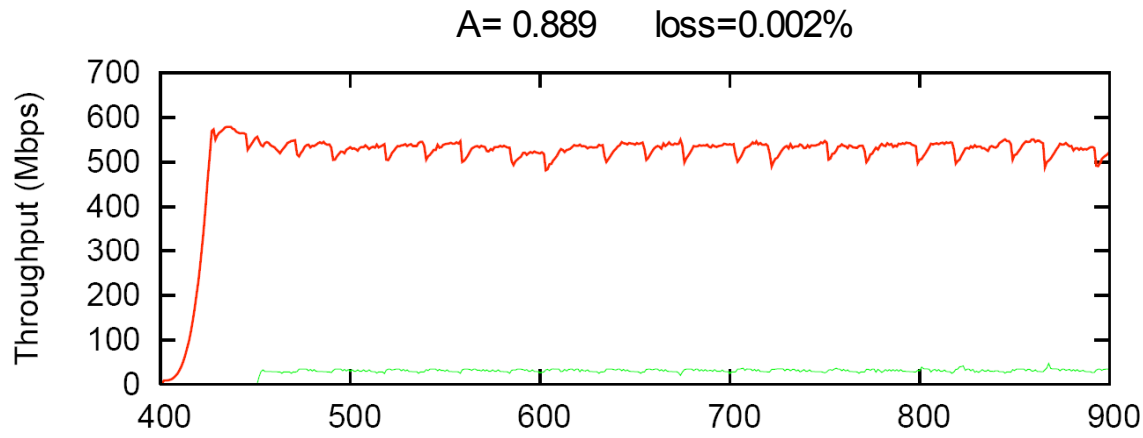

Figure 18: CUBIC – FAST Simulation with Background Traffic

Figure 17 and Figure 18 show the graphs of the CUBIC – FAST simulation with and without background traffic respectively. In these figures the red (darker) line represents CUBIC flow (flow 1) and green (lighter) line represents FAST flow (flow2).

In this case, the fairness in sharing the bandwidth reduces with the addition of background traffic. This is because FAST considers queuing delay as the primary measure of congestion. With the addition of background traffic, queuing delay increases and as a result FAST becomes less aggressive. We note that in all cases that involved FAST, the FAST flow suffered when background traffic was added.

5. Conclusion

We simulated a total of 16 combinations of experiments with HS-TCP, Scalable, CUBIC and FAST along with adding background traffic. We expected some significant changes in the fairness of sharing of the bandwidth. As expected we observed changes, some of which are summarized below:

- Scalable TCP which follows static approach in adjusting the congestion window became fairer with the addition of background traffic.
- FAST which is a delay based protocol became less aggressive as queuing delay is increased with the addition of background traffic.
- CUBIC and HS-TCP have become more fair in sharing the bandwidth when competing against each other.

References

[1] M.C. Weigle, P. Sharma, and J. Freeman, Performance of Competing High-Speed TCP Flows, *Proceedings of NETWORKING*, Coimbra, Portugal, May 2006, pp. 476-487.

[2] Pankaj Sharma, MS Thesis: Performance Analysis of High-Speed Transport Control Protocols, Aug 2006.

[3] Prashanth Adurthi, MS Thesis: Generating Tmix-Based TCP Application Workloads in ns-2 and GTNetS, Aug 2006.

[4] S. Hegde, D. Lapsley, B.Wydrowski, J. Lindheim, D.Wei, C. Jin, S. Low, H.Newman, "FAST TCP in High Speed Networks: An Experimental Study", Proceedings GridNets, San Jose, 2004.

[5] T. Kelly. Scalable TCP: Improving performance in high speed wide area networks. In Proceedings of PFLDnet, Geneva, Switzerland, Feb. 2003.

[6] I. Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In Proceedings of PFLDnet, Lyon, France, Feb. 2005.

[7] S.Floyd. “High-Speed TCP for Large Congestion Windows”. IETF RFC 3649, December 2003.

[8] I. Rhee and L. Xu. Simulation code and scripts for CUBIC, 2005
http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-script/script.htm

[9] T. Cui and L. Andrew. FAST TCP simulator module for ns-2, version 1.1, 2004
http://www.cubinlab.ee.mu.oz.au/ns2fasttcp

[10] M.C. Weigle, P. Adurthi, F. Hernandez-Campos, K. Jeffay, and F. D. Smith, Tmix: a tool for generating realistic TCP application workloads in ns-2, *ACM SIGCOMM Computer Communication Review*, Volume 36, Issue 3 (July 2006),

Appendix A

```
# tmix-lossless.tcl
#
# Simulation script to simulate the tmix-ns component
#
# useful variables
set length 910;             # length of traced simulation (s)  (45m)
#set length 3;
set warmup 0;                # warmup interval (s)
set end $length
#set debug 5

set DATADIR "."
set FILE "tmix-ns-Abilene-lossless"
set CVECDIR "tcvecs"
set INBOUND "$CVECDIR/Abilene_sorted_crecv.ns"
set OUTBOUND "$CVECDIR/Abilene_sorted_cinit.ns"
set BINDIR "bin"


#-------------------------------------------HSTCP STARTS-----------------------
-------------------------------------------------------

# ARGS: endtime  flow1_type  flow2_type  FASTalpha  flow2_start run  DATADIR

set begintime  [clock second]

# simulation end time
set endtime [lindex $argv 0]

# high-speed flow 1 type, flow 2 type
set hstcp_type(0) [lindex $argv 1];   # HS, Scalable, FAST, BIC, CUBIC
set hstcp_type(1) [lindex $argv 2];   # HS, Scalable, FAST, BIC, CUBIC
set hstcpflows_num 2
set hstcpflows_type 0

# FASTalpha
set alpha [lindex $argv 3]

# flow 2 start time
set flow2_start [lindex $argv 4]

# run
set run [lindex $argv 5]

# datadir
set DATADIR1 [lindex $argv 6]

set bandwidth 622;   # 622 Mbps
set delay 48;        # total of 100 ms RTT
set bf_size 1555;    # qlen is 20% BDP

# parameters for CUBIC
set cubic_tcp_mode 1

set low_window 0
set high_window 83000
set high_p 0.0000001
```

```
set high_decrease 0.1
set hstcp_fix 1

remove-all-packet-headers       ; # removes all except common
add-packet-header Flags IP TCP  ; # hdrs reqd for TCP

source utils.tcl

set frep [open $DATADIR1/report.txt w]
set urep [open $DATADIR1/util.out w]

puts "-------------------------------------------------------------"
puts "-------------------------------------------------------------"
puts [pwd]

printlegend

#--------------------------------------------HSTCP ENDS-------------------------
-------------------------------------------------------

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Setup Simulator
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

remove-all-packet-headers;             # removes all packet headers
add-packet-header IP TCP;              # adds TCP/IP headers
set ns [new Simulator];                # instantiate the Simulator
$ns use-scheduler Heap



#-----------------------------------------HSTCP STARTS-------------------------
-------------------------------------------------------

global defaultRNG
$defaultRNG seed 9999;

for {set i 1} {$i < $run} {incr i} {
    $defaultRNG next-substream
}

Agent/TCP set timestamps_ 1
Agent/TCP set window_ 67000
Agent/TCP set packetSize_ 1000
Agent/TCP set overhead_ 0.000008
Agent/TCP set max_ssthresh_ 100
Agent/TCP set maxburst_ 2

# BIC
Agent/TCP set bic_beta_ 0.8
Agent/TCP set bic_B_ 4
Agent/TCP set bic_max_increment_ 32
Agent/TCP set bic_min_increment_ 0.01
Agent/TCP set bic_fast_convergence_ 1
Agent/TCP set bic_low_utilization_threshold_ 0
Agent/TCP set bic_low_utilization_checking_period_ 2
Agent/TCP set bic_delay_min_ 0
Agent/TCP set bic_delay_avg_ 0
Agent/TCP set bic_delay_max_ 0
Agent/TCP set bic_low_utilization_indication_ 0

# CUBIC
Agent/TCP set cubic_beta_ 0.8
```

```
Agent/TCP set cubic_max_increment_ 16
Agent/TCP set cubic_fast_convergence_ 1
Agent/TCP set cubic_scale_ 0.4
Agent/TCP set cubic_tcp_friendliness_ $cubic_tcp_mode
Agent/TCP set cubic_low_utilization_threshold_ 0
Agent/TCP set cubic_low_utilization_checking_period_ 2
Agent/TCP set cubic_delay_min_ 0
Agent/TCP set cubic_delay_avg_ 0
Agent/TCP set cubic_delay_max_ 0
Agent/TCP set cubic_low_utilization_indication_ 0

puts "BufferSize(Packets) $bf_size"
puts $frep "BufferSize(Packets) $bf_size"

set starttime(0) 400
set starttime(1) $flow2_start


#-----------------------------------------HSTCP ENDS---------------------------
-------------------------------------------------------
```

```
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Setup Topology
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

#           $bw Mb
#  client  --------  server
#  cloud     0ms     cloud
#
#   n(0)              n(1)

# create nodes
set n(0) [$ns node]
set n(1) [$ns node]
set n(2) [$ns node]
set n(3) [$ns node]

# setup TCP
Agent/TCP/FullTcp set segsize_ 1000;           # set MSS to 1460 bytes
#Agent/TCP/FullTcp set nodelay_ true;           #disabling nagle
#Agent/TCP/FullTcp set segsperack_ 2;           #delayed ACKs
#Agent/TCP/FullTcp set interval_ 0.1;           # 100 ms

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Setup TmixNode
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

set tmix(0) [new Tmix_End]
$tmix(0) set-init $n(0);                 # name $n(0) as initiator
$tmix(0) set-acc $n(1);                  # name $n(1) as acceptor
$tmix(0) set-ID 7
$tmix(0) set-cvfile "$INBOUND"

set tmix(1) [new Tmix_End]
$tmix(1) set-init $n(3);                 # name $n(3) as initiator
$tmix(1) set-acc $n(2);                  # name $n(2) as acceptor
$tmix(1) set-ID 8
$tmix(1) set-cvfile "$OUTBOUND"
```

```
#
# Setup tmixDelayBox
#
set tmixNet(0) [$ns tmix_net]
$tmixNet(0) set-flowfile "$INBOUND" [$n(0) id] [$n(1) id]
$tmixNet(0) set-lossless

set tmixNet(1) [$ns tmix_net]
$tmixNet(1) set-flowfile "$OUTBOUND" [$n(3) id] [$n(2) id]
$tmixNet(1) set-lossless

#$tmix(0) set-debug $debug
#$tmix(1) set-debug $debug
#$tmixNet(0) set-debug $debug
#$tmixNet(1) set-debug $debug

# create link
$ns duplex-link $n(0) $tmixNet(0) 1000Mb 1ms DropTail
$ns duplex-link $n(2) $tmixNet(0) 1000Mb 1ms DropTail
$ns duplex-link $tmixNet(0) $tmixNet(1) 622Mb 48ms DropTail
$ns duplex-link $tmixNet(1) $n(1) 1000Mb 1ms DropTail
$ns duplex-link $tmixNet(1) $n(3) 1000Mb 1ms DropTail

# set queue buffer sizes (in packets)  (default is 20 packets)
$ns queue-limit $n(0) $tmixNet(0) 67000
$ns queue-limit $tmixNet(0) $n(0) 67000
$ns queue-limit $n(2) $tmixNet(0) 67000
$ns queue-limit $tmixNet(0) $n(2) 67000
$ns queue-limit $tmixNet(0) $tmixNet(1) 1555
$ns queue-limit $tmixNet(1) $tmixNet(0) 1555
$ns queue-limit $tmixNet(1) $n(1) 67000
$ns queue-limit $n(1) $tmixNet(1) 67000
$ns queue-limit $tmixNet(1) $n(3) 67000
$ns queue-limit $n(3) $tmixNet(1) 67000

#------------------------HSTCP STARTS-----------------------------------------
-------------------------

set qmon [$ns monitor-queue $tmixNet(0) $tmixNet(1) ""]
set qfp [open $DATADIR1/queue.out w]
print-queue 0.1 $qfp

set fmon [$ns makeflowmon Fid]
$ns attach-fmon [$ns link $tmixNet(0) $tmixNet(1)] $fmon

# back path fmon
set fmon2 [$ns makeflowmon Fid]
$ns attach-fmon [$ns link $tmixNet(1) $tmixNet(0)] $fmon2


for {set i 0} {$i < $hstcpflows_num} {incr i} {
  set hsnode(s$i) [$ns node]
  set hsnode(r$i) [$ns node]

  $ns duplex-link $hsnode(s$i) $tmixNet(0) 1000Mb 1ms DropTail
  $ns duplex-link $hsnode(r$i) $tmixNet(1) 1000Mb 1ms DropTail
  puts "hstcp$i, forwardlink delay=1 ms"
  puts $frep "hstcp$i, forwardlink delay=1 ms"

  puts "hstcp$i, RTT = 100.00 ms"
  puts $frep "hstcp$i, RTT = 100.00 ms"
```

```
  $ns queue-limit $hsnode(s$i) $tmixNet(0) 67000
  $ns queue-limit $tmixNet(0) $hsnode(s$i) 67000

  $ns queue-limit $tmixNet(1) $hsnode(r$i) 67000
  $ns queue-limit $hsnode(r$i) $tmixNet(1) 67000

    if {$hstcp_type($i) != "FAST"} {
       set hstcp$i [$ns create-connection TCP/Sack1 $hsnode(s$i) TCPSink/Sack1
$hsnode(r$i) $i]

       if {$hstcp_type($i) == "HS"} {
           set hstcpflows_type 8
           set low_window 38
       } elseif {$hstcp_type($i) == "Scalable"} {
           set hstcpflows_type 9
           set low_window 16
           [set hstcp$i] set scalable_lwnd_ $low_window
       } elseif {$hstcp_type($i) == "BIC"} {
           set hstcpflows_type 12
           set low_window 14
       } elseif {$hstcp_type($i) == "CUBIC"} {
           set hstcpflows_type 13
       } elseif {$hstcp_type($i) == "HTCP"} {
            set hstcpflows_type -10
        }

if {$hstcp_type($i) != "HTCP"} {
        [set hstcp$i] set max_ssthresh_ 100
} else  {
        [set hstcp$i] set max_ssthresh_ 1
}

       [set hstcp$i] set windowOption_ $hstcpflows_type
       [set hstcp$i] set low_window_ $low_window
       [set hstcp$i] set high_window_ $high_window
       [set hstcp$i] set high_p_ $high_p
       [set hstcp$i] set high_decrease_ $high_decrease
       [set hstcp$i] set hstcp_fix_ $hstcp_fix
    } else {
       set hstcp$i [$ns create-connection TCP/Fast $hsnode(s$i) TCPSink/Sack1
$hsnode(r$i) $i]
        [set hstcp$i] set alpha_ $alpha
        [set hstcp$i] set beta_ $alpha
    }

  set hsftp$i [[set hstcp$i] attach-app FTP]

  set lastBytes($i) 0

  set cfp_hs($i) [open $DATADIR1/cwnd_hstcp$i.out w]
  set tput_hs($i) [open $DATADIR1/tput_hstcp$i.out w]
  if { $hstcpflows_type == 12 } {
    print-cwnd-bic [set hstcp$i] 0.1 $cfp_hs($i)
  } else {
    print-cwnd [set hstcp$i] 0.1 $cfp_hs($i)
  }

  set lastKBytes($i) 0
  print-th-one $i $fmon 1
}
```

```
#------------------------HSTCP ENDS-------------------------------------------
-------------------------

# record statistics
#$tmix(0) set-outfile "$DATADIR/${FILE}-1.dat"
#$tmix(1) set-outfile "$DATADIR/${FILE}-2.dat"

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Packet Tracing
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

proc trace {} {
    global ns n tmixNet DATADIR FILE hstcpflows_num

    # setup packet tracing
    Trace set show_tcphdr_ 1
    set qmonf [open "$DATADIR/${FILE}.dat" w]
    $ns trace-queue $n(0) $tmixNet(0) $qmonf
    $ns trace-queue $tmixNet(0) $n(0) $qmonf
    $ns trace-queue $tmixNet(1) $n(1) $qmonf
    $ns trace-queue $n(1) $tmixNet(1) $qmonf
    $ns trace-queue $tmixNet(0) $tmixNet(1) $qmonf
    $ns trace-queue $tmixNet(1) $tmixNet(0) $qmonf
    $ns trace-queue $n(2) $tmixNet(0) $qmonf
    $ns trace-queue $tmixNet(0) $n(2) $qmonf
    $ns trace-queue $tmixNet(1) $n(3) $qmonf
    $ns trace-queue $n(3) $tmixNet(1) $qmonf
    for {set i 0} {$i < $hstcpflows_num} {incr i} {
    global hsnode tmixNet
    $ns trace-queue $hsnode(s$i) $tmixNet(0) $qmonf
    $ns trace-queue $tmixNet(0) $hsnode(s$i) $qmonf
    $ns trace-queue $hsnode(r$i) $tmixNet(1) $qmonf
    $ns trace-queue $tmixNet(1) $hsnode(r$i) $qmonf
    }

}

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Cleanup
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

proc finish {} {
    global ns
    $ns flush-trace

}

#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Simulation Schedule
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

$ns at 0.0 "$tmix(0) start"
$ns at 0.0 "$tmix(1) start"
$ns at $warmup "trace"
for {set i 0} {$i < $hstcpflows_num} {incr i} {
global starttime hsftp
$ns at $starttime($i) "[set hsftp$i] start"
# print this connection start time
  puts "hstcp$i starttime: $starttime($i)"
  puts $frep "hstcp$i starttime: $starttime($i)"
```

```
}

$ns at $end "$tmix(0) stop"
$ns at $end "$tmix(1) stop"
for {set i 0} {$i < $hstcpflows_num} {incr i} {
global endtime hsftp
exec echo $endtime > temp2
$ns at [expr $endtime] "[set hsftp$i] stop"
}


# keep track of real time, sim time, and memory
set time_mem $DATADIR/exp-TIME-MEM
exec touch $time_mem;     exec rm $time_mem;     exec touch $time_mem
exec printf "\#sim    real    mem\n" >> $time_mem
set USER [exec whoami]
for {set j 0} {$j <= $length} {incr j} {
    $ns at $j "exec printf \"$j\" >> $time_mem; \ exec $BINDIR/format-time-
mem.pl $USER >> $time_mem"
}

set halftime [expr $endtime / 2]

puts stderr "halftime: $halftime   endtime: $endtime"

for {set i 0} {$i < $hstcpflows_num} {incr i} {
  $ns at $halftime "print-stat-one-pre $i $fmon"
  $ns at $endtime "print-stat-one $i $fmon"

}

$ns at $halftime "print-stat-all-pre"
$ns at $endtime "print-stat-all"

$ns at $endtime "printlegend"
$ns at [expr $length] "finish"
$ns at [expr $length + 1] "finish1"

$ns at 0 "timeReport 1"
$ns at 0.1 "print-util 0.1 0"

#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Start the Simulation
#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

$ns run
```

Appendix B

Scalable – FAST Simulation Graphs

A = 0.488 loss = 0.043%

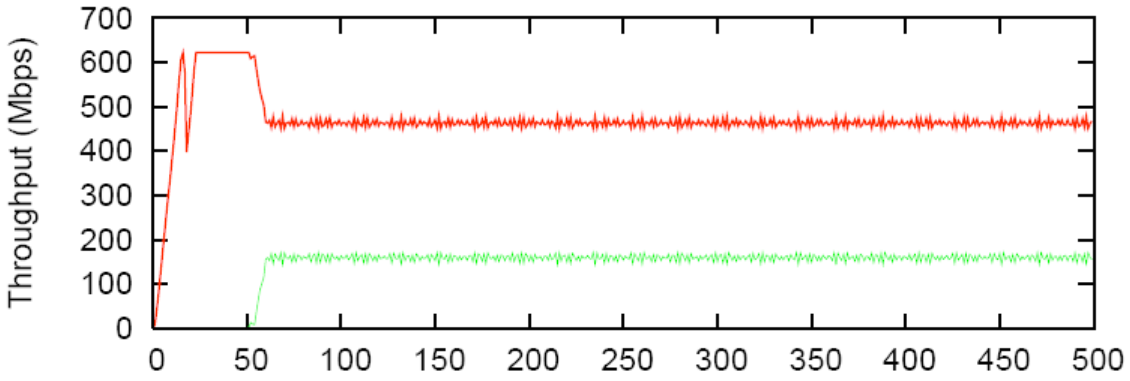

Figure 19: Scalable – FAST Simulation without Background Traffic

A = 0.932 loss = 0.153%

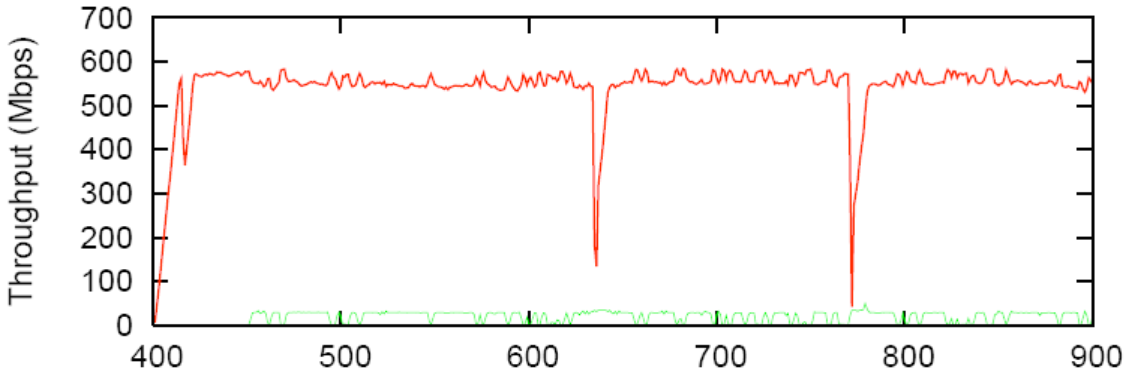

Figure 20: Scalable – FAST Simulation with Background Traffic

Scalable – CUBIC Simulation Graphs

A = 0.987 loss = 0.133%

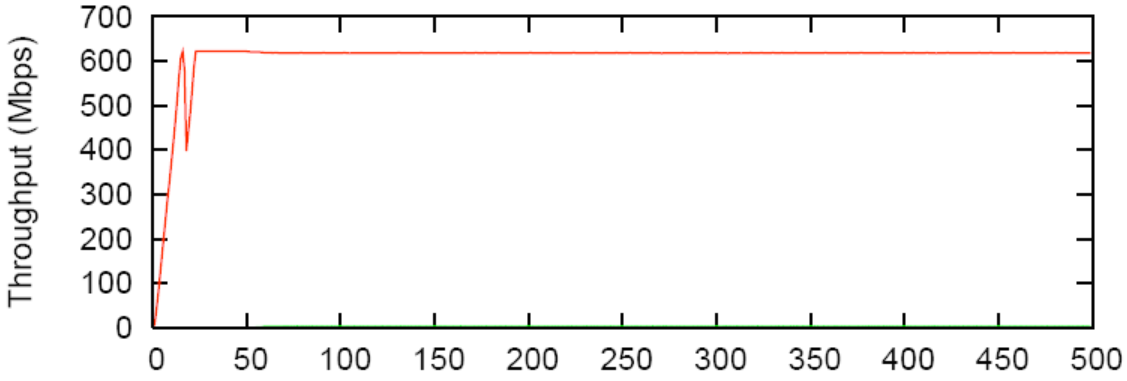

Figure 21: Scalable – CUBIC Simulation without Background Traffic

A = 0.857 loss = 0.118%

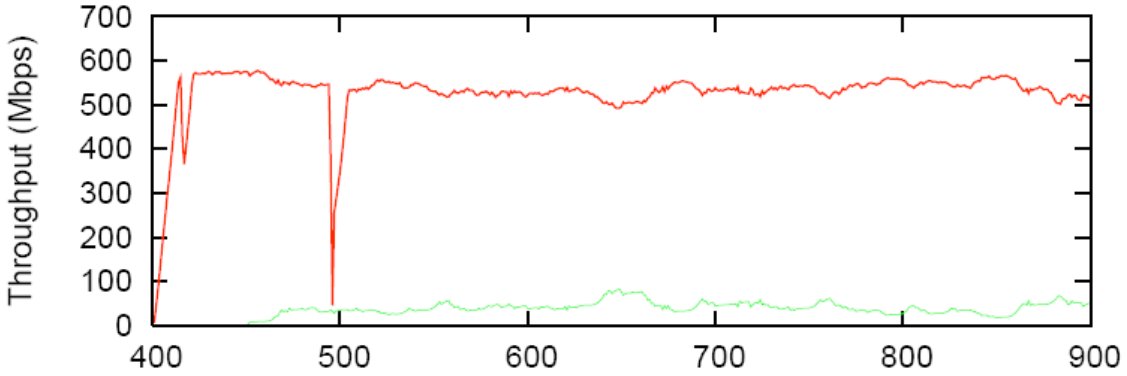

Figure 22: Scalable – CUBIC Simulation with Background Traffic

HS – Scalable Simulation Graphs

A = -0.996 loss = 0.079%

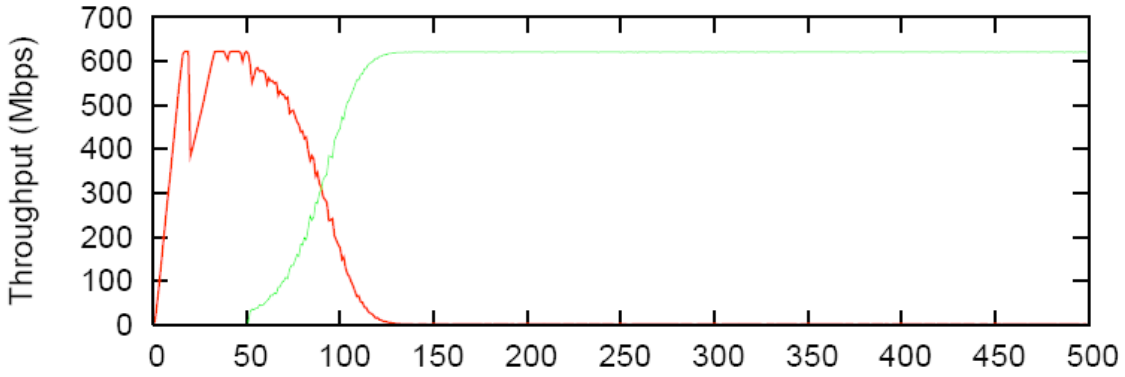

Figure 23: HS - Scalable Simulation without Background Traffic

A = -0.803 loss = 0.09%

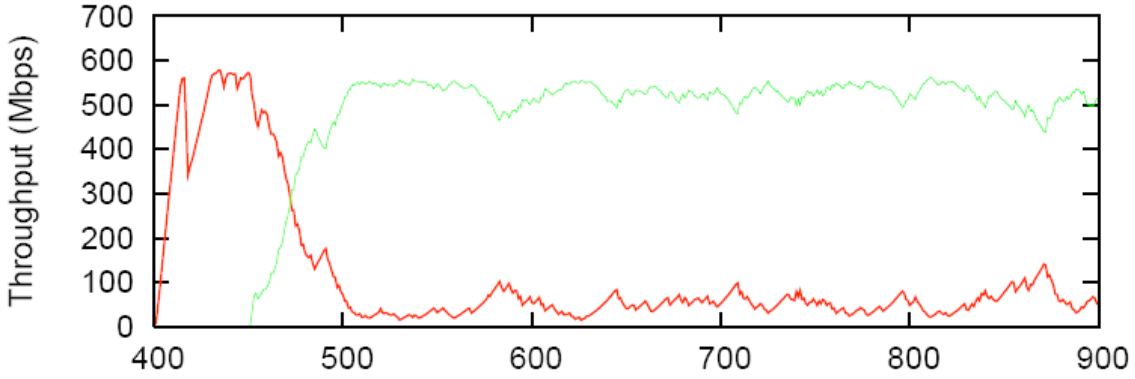

Figure 24: HS - Scalable Simulation with Background Traffic

HS – FAST Simulation Graphs

A = 0.451 loss = 0.004%

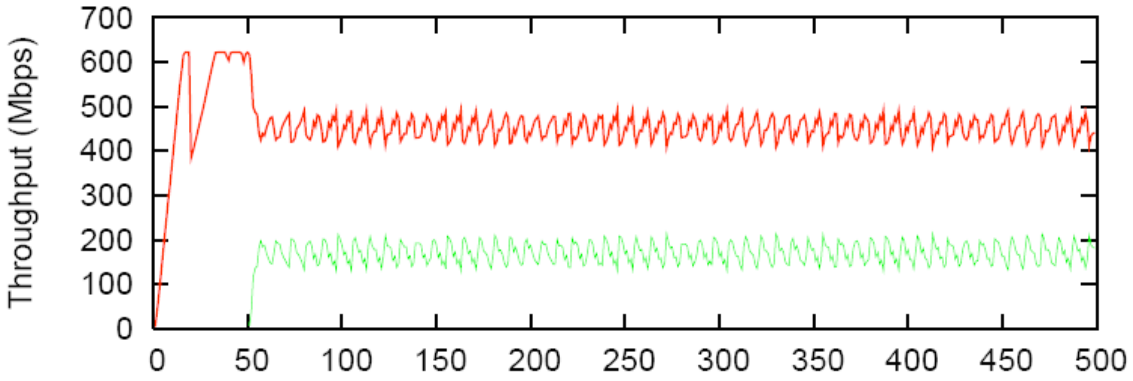

Figure 25: HS - FAST Simulation without Background Traffic

A = 0.891 loss = 0.01%

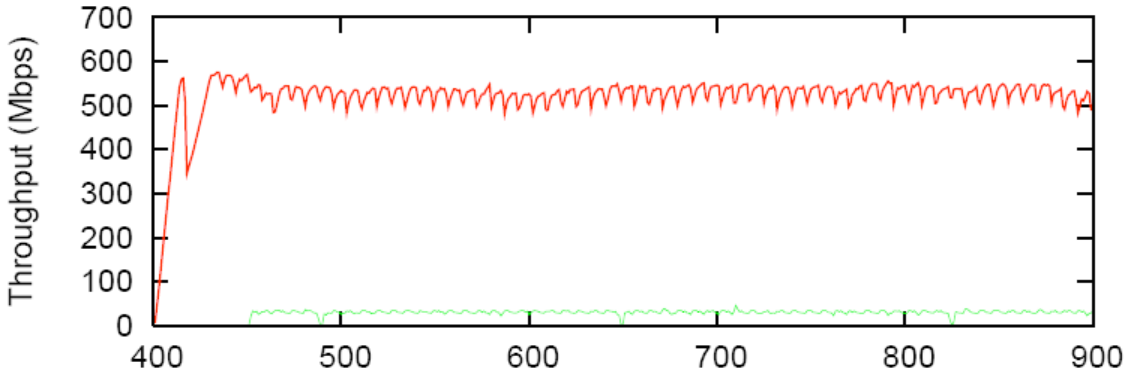

Figure 26: HS - FAST Simulation with Background Traffic

HS – HS Simulation Graphs

A = 0.074 loss = 0.015%

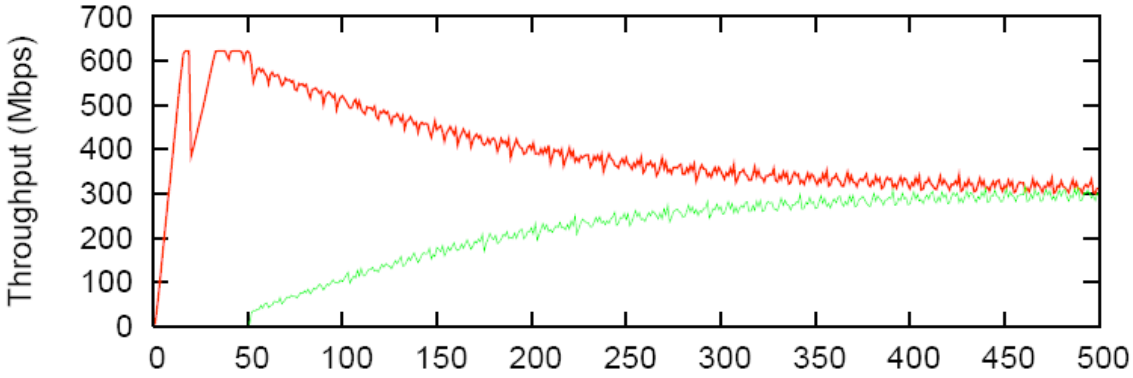

Figure 27: HS - HS Simulation without Background Traffic

A = -0.123 loss = 0.024%

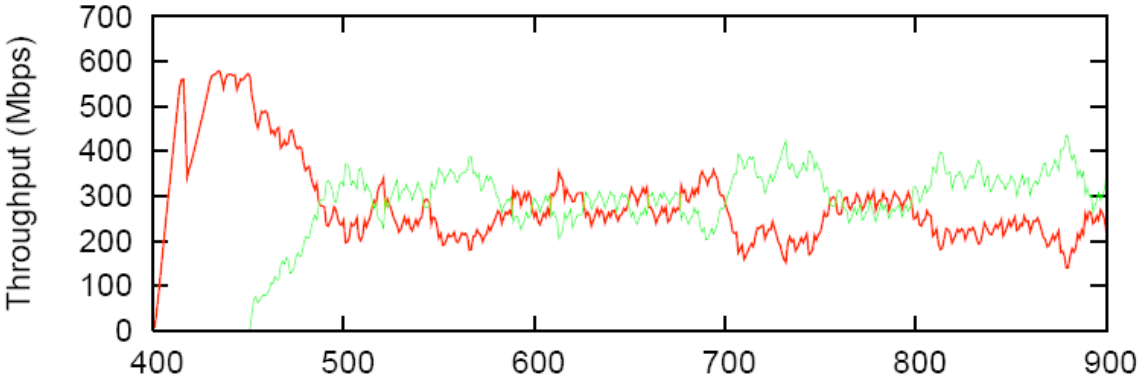

Figure 28: HS - HS Simulation with Background Traffic

FAST – Scalable Simulation Graphs

A = -0.489 loss = 0.033%

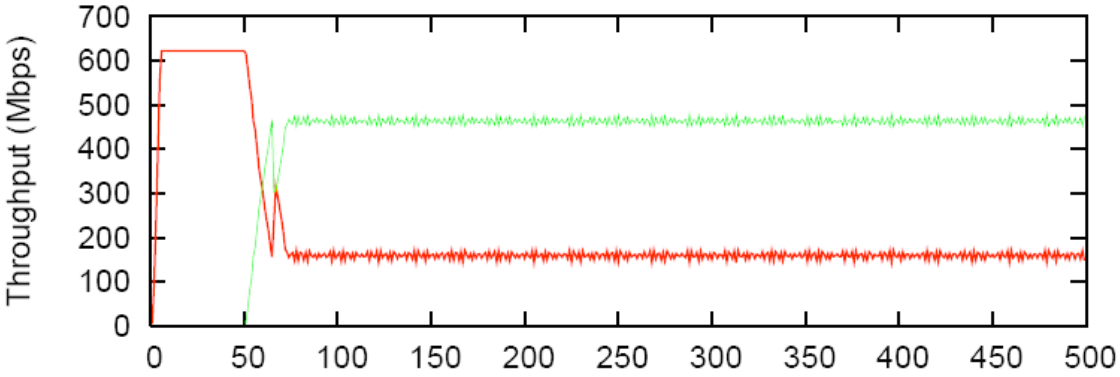

Figure 29: FAST - Scalable Simulation without Background Traffic

A = -0.921 loss = 0.127%

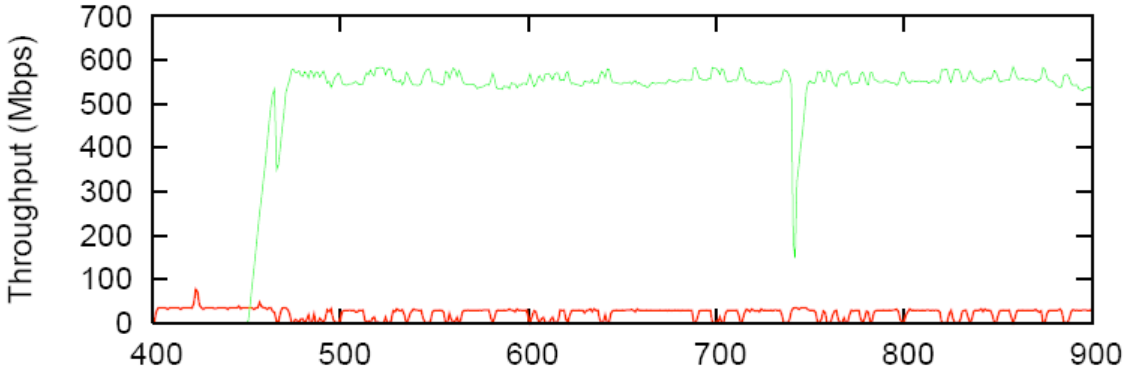

Figure 30: FAST - Scalable Simulation with Background Traffic

FAST – HS Simulation Graphs

A = -0.452 loss = 0.003%

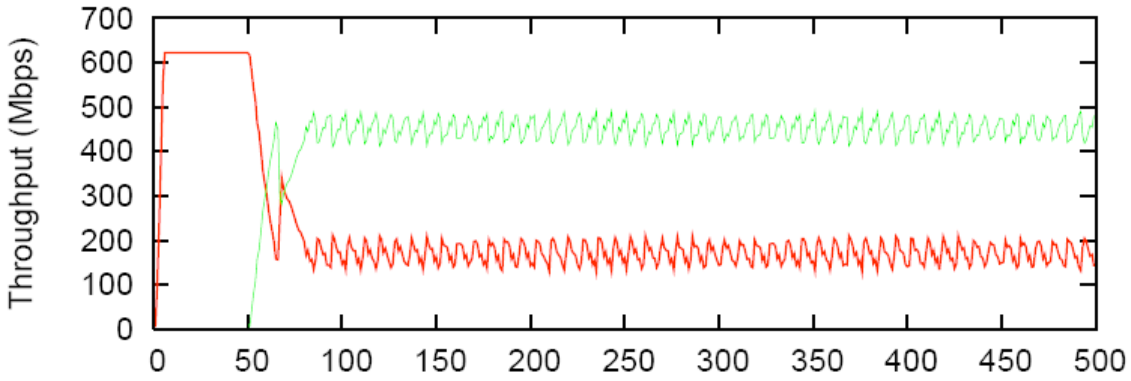

Figure 31: FAST - HS Simulation without Background Traffic

A = -0.891 loss = 0.007%

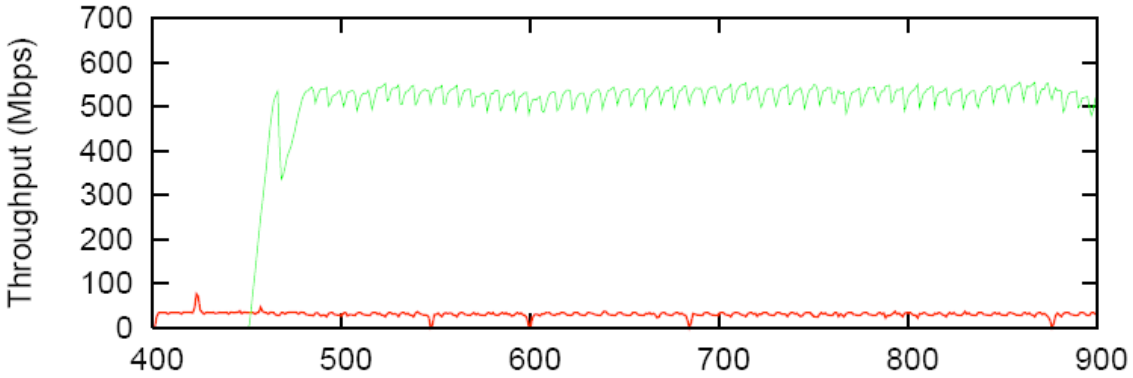

Figure 32: FAST - HS Simulation with Background Traffic

FAST – FAST Simulation Graphs

A = -0.233 loss = 0.000%

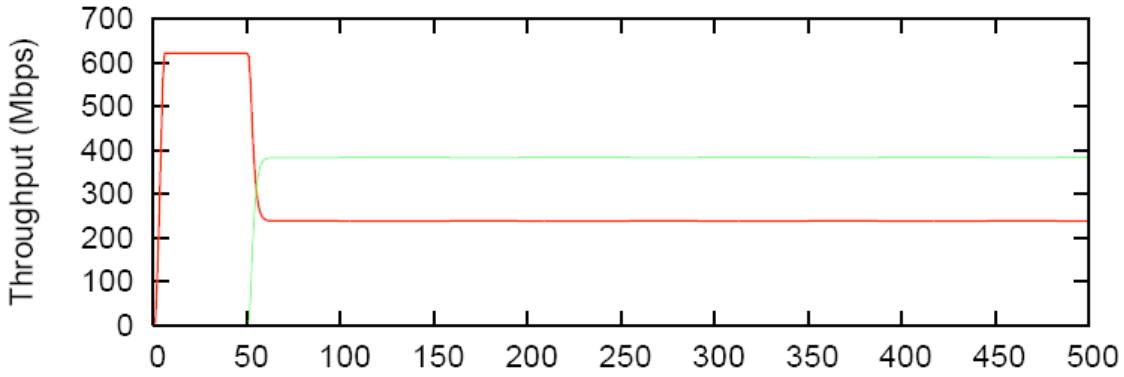

Figure 33: FAST – FAST Simulation without Background Traffic

A = 0.039 loss = 0.000%

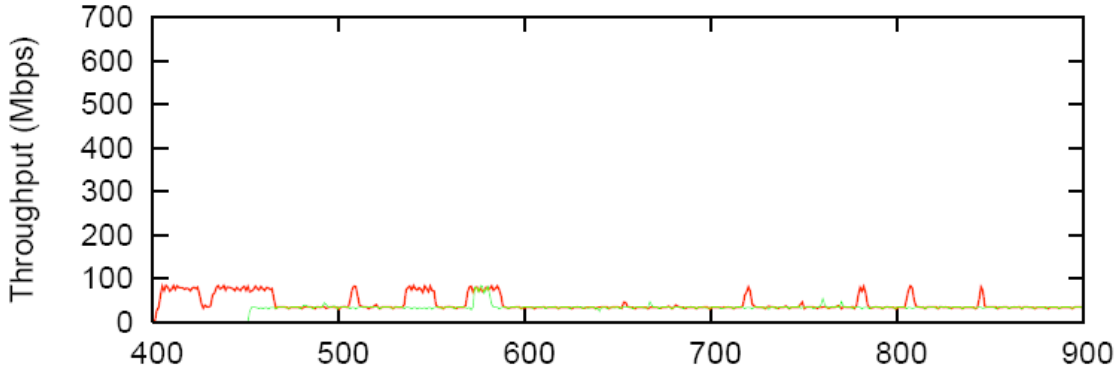

Figure 34: FAST – FAST Simulation with Background Traffic

FAST – CUBIC Simulation Graphs

A = -0.487 loss = 0.001%

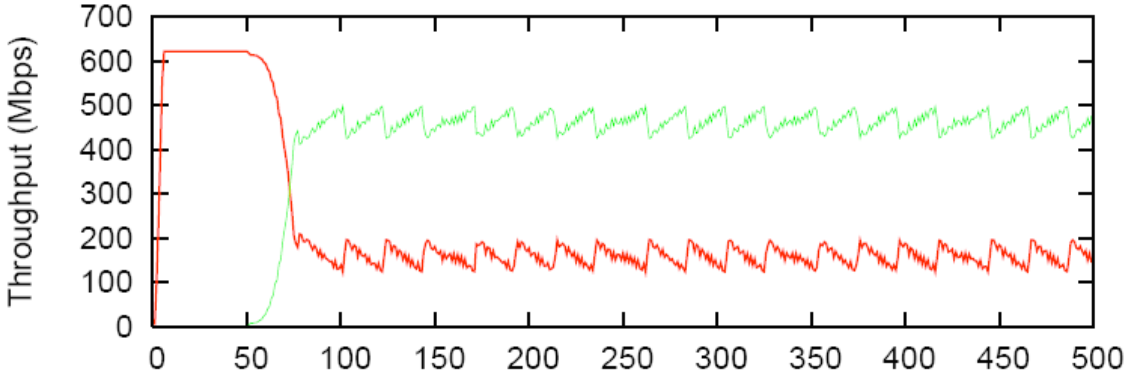

Figure 35: FAST - CUBIC Simulation without Background Traffic

A = -0.89 loss = 0.002%

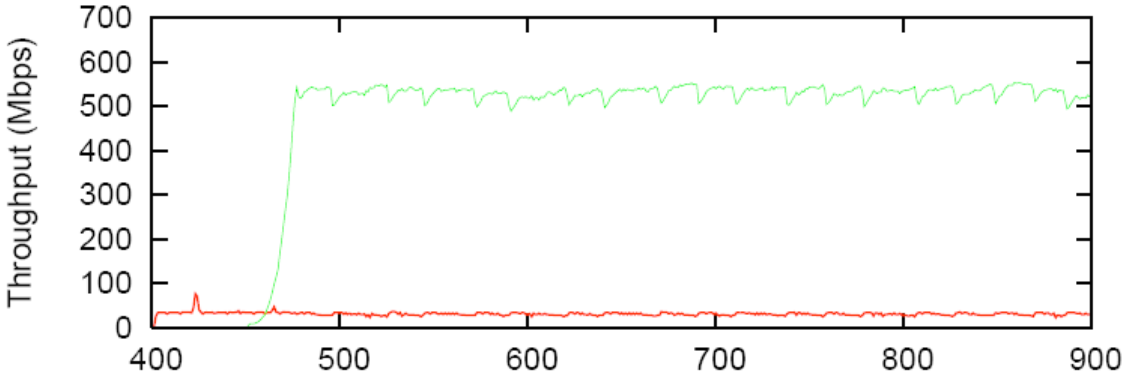

Figure 36: FAST - CUBIC Simulation with Background Traffic

CUBIC – Scalable Simulation Graphs

A = -0.978 loss = 0.073%

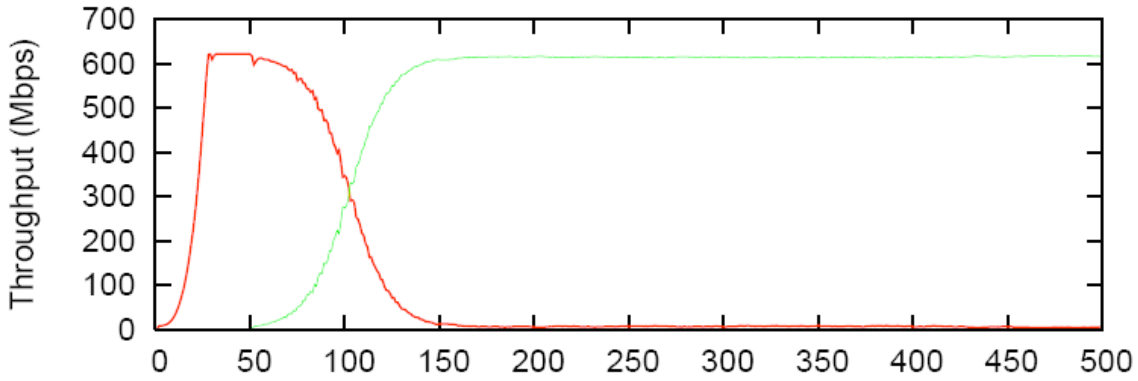

Figure 37: CUBIC - Scalable Simulation without Background Traffic

A = -0.877 loss = 0.09%

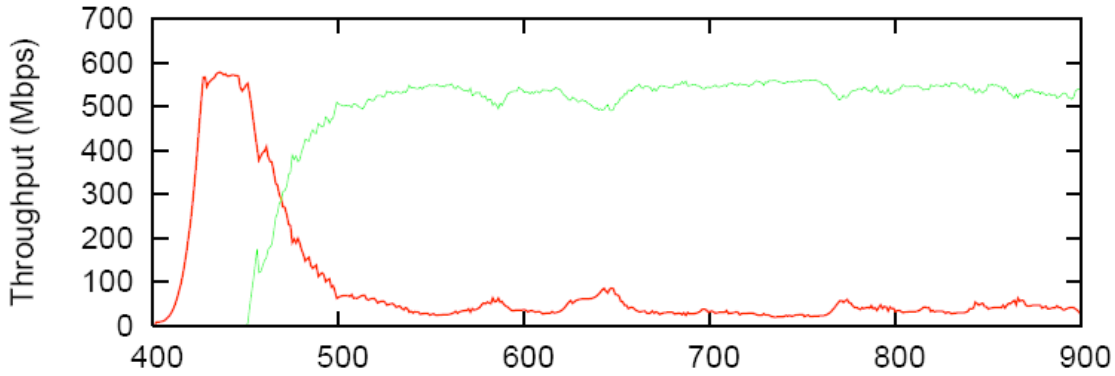

Figure 38: CUBIC - Scalable Simulation with Background Traffic

CUBIC – HS Simulation Graphs

A = -0.481 loss = 0.008%

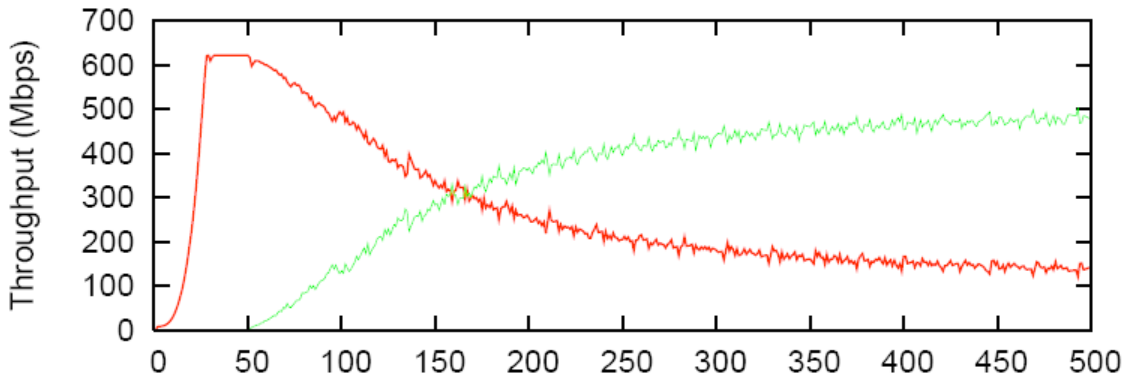

Figure 39: CUBIC - HS Simulation without Background Traffic

A = -0.358 loss = 0.018%

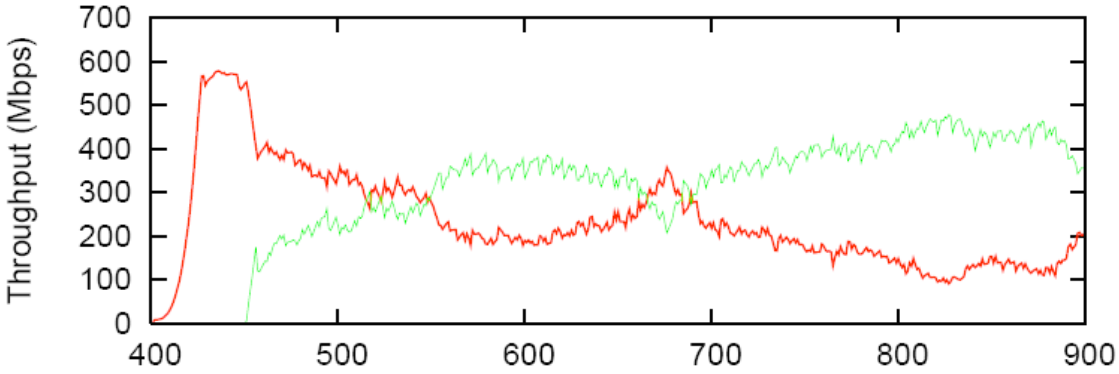

Figure 40: CUBIC - HS Simulation with Background Traffic

CUBIC – CUBIC Simulation Graphs

A = 0.08 loss = 0.003%

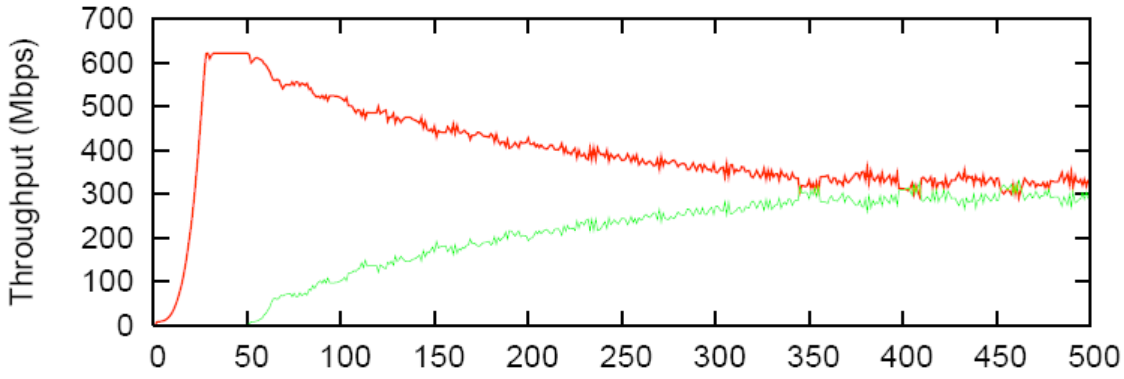

Figure 41: CUBIC - CUBIC Simulation without Background Traffic

A = -0.031 loss = 0.009%

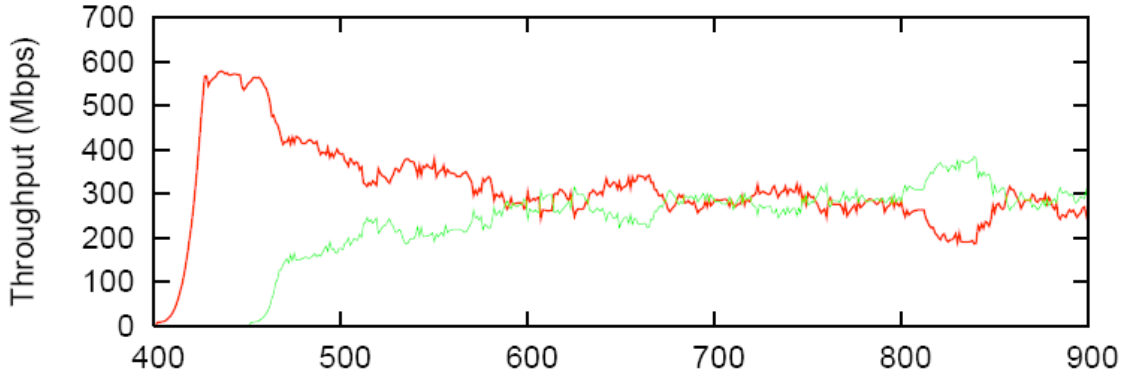

Figure 42: CUBIC - CUBIC Simulation with Background Traffic