High Speed TCP

Comp 6320

Adam Brown
Ben Gustafson

Table of Contents

Introduction

Over 95% of all Internet traffic uses TCP as a transport protocol. The protocol includes congestion control algorithms written by Van Jacobson. Over the years TCP has been tuned for the most common applications and environments. Combine these factors and one could make a solid argument for TCP's congestion control algorithms being the reason the current Internet infrastructure isn't continuously over run with packets.

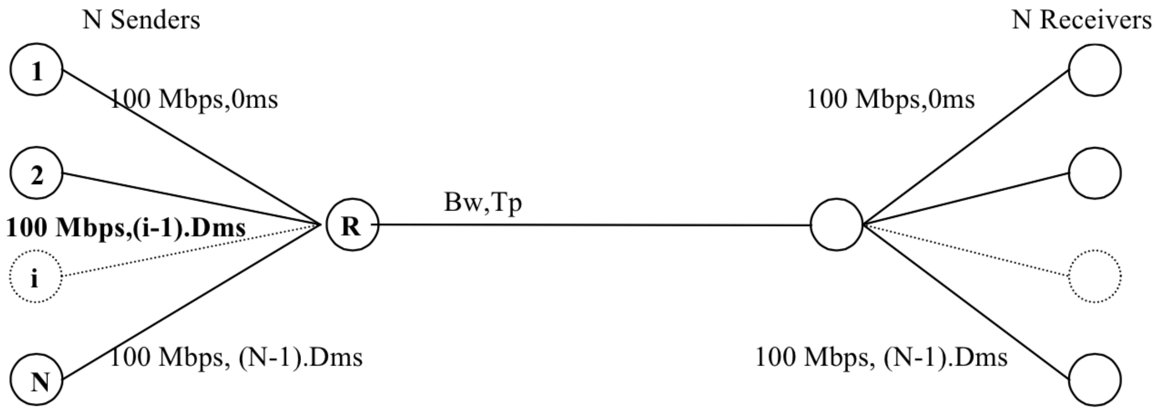
However, with the ever-increasing speeds the current version of TCP does not perform well. The congestion control mechanizes cap the maximum amount of data one instance can send by limiting the congestion window. In 2003 a new version of TCP was introduced: High Speed TCP.

High Speed TCP was first proposed by Sally Floyd via RFC 3649. She introduced the idea of removing the cap on the congestion window when the bandwidth allowed. Floyd saw the need for a new protocol to cover the increasingly common case of a high-speed connection.

In this study we seek to find out the differences between TCP (we will be using TCP NewReno) and High Speed TCP and effectiveness of High Speed TCP. We will find if High Speed TCP is tuned for high-speed environments as well as find the best environment for High Speed TCP.

Methodology

To thoroughly test each protocol several different factors of the network have to be changed. We chose to change bandwidth (Bw), propagation time (Tp), network delay (D), the number of senders / receivers. (N)



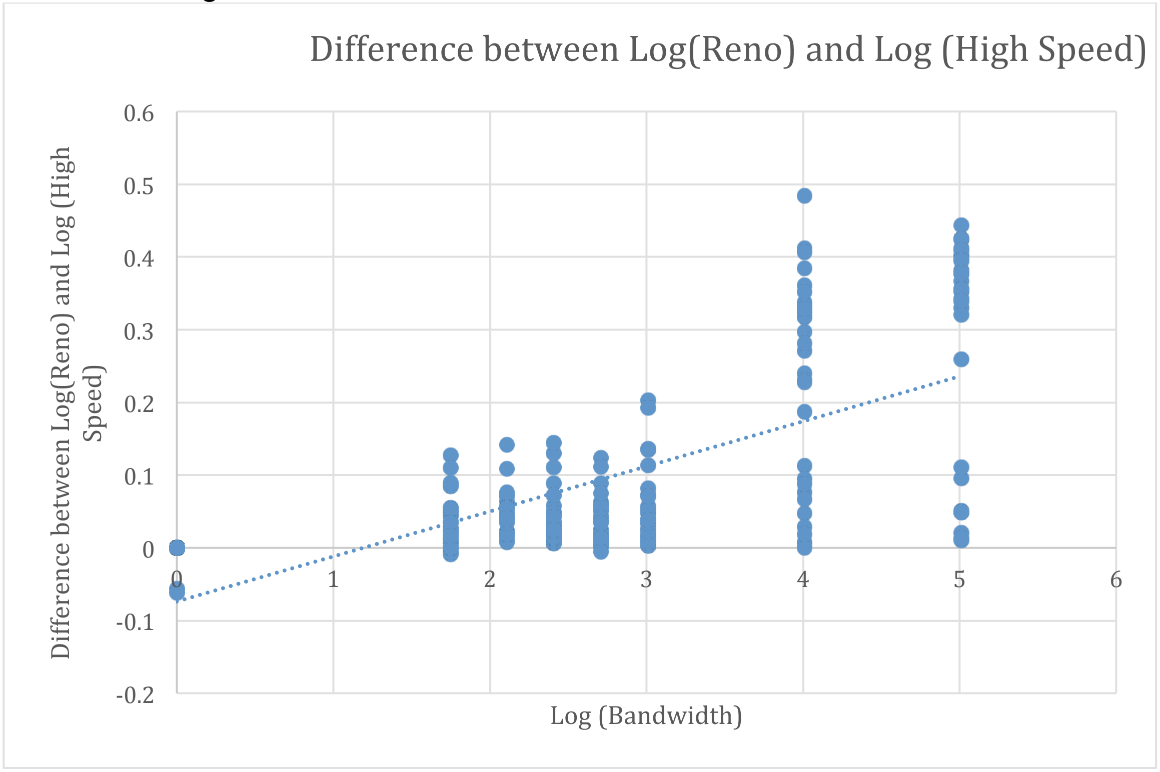
We used 1kbps, 56kbps, 128kbps, 256kbps, 512kbps, 1Mbps, 10Mbps, 100Mbps as our bandwidth bench marks. Along with propagation times of 50ms, 250ms, 500ms, and 1s. Delay alternated between the values of 0 and 100, while we used sets of 2, 8, 16, and 32 senders and receivers.

Each experiment consisted of a bandwidth, propagation time, delay value, a number of senders and a type of TCP. 30 different simulations of each experiment were run using NS-2, for statistically valid results. The average throughput of all senders and the average fairness of each node was collected and graphed below.

Results

The most common metric used to measure the effectiveness of each protocol is the difference between the log of the throughput of TCP NewReno and the log of the High Speed TCP throughput. The resulting amount is the log throughput that High Speed TCP sent more than TCP NewReno. Taking the log of the throughput allows for the scale of the axis to be small enough to fit all cases on the same graph.

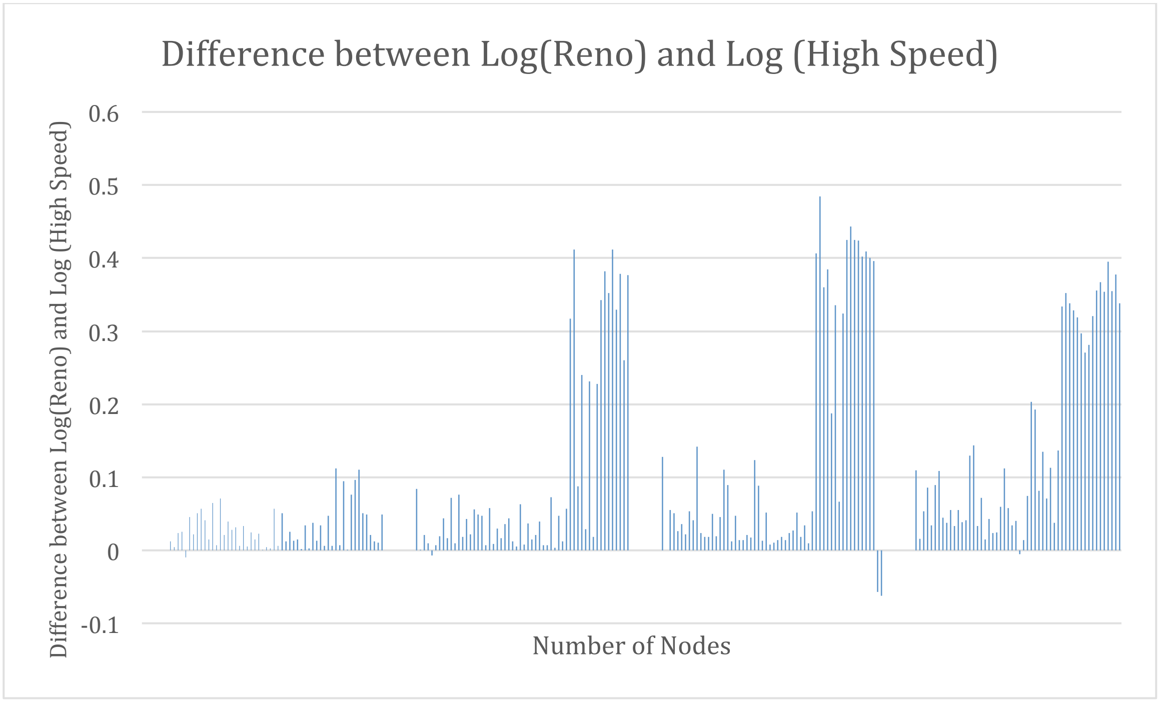
Below the trend-line increases as bandwidth increases. Thus High Speed TCP on average sends more data than TCP NewReno.



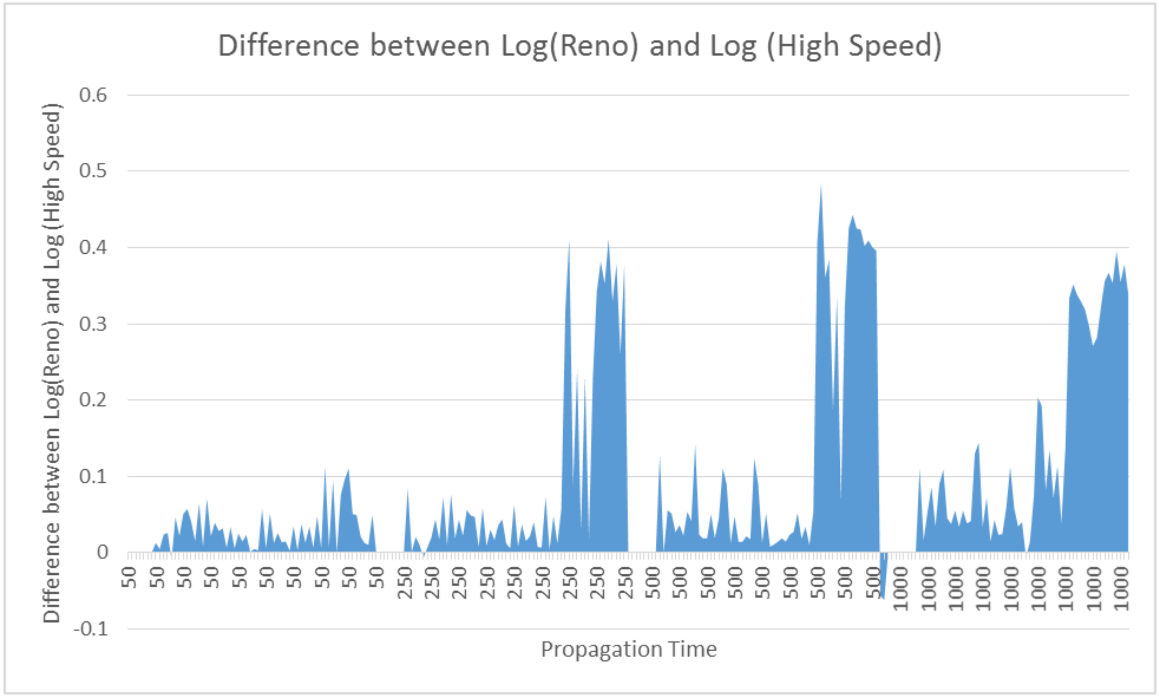
Let us explore what causes the best-case scenario and the worst case scenario for High Speed TCP.

The graph below is grouped based on the number of nodes during the experiment with the group of 2 nodes on the left and the group of 32 nodes on the right. We can see as the number of nodes increase, the better High Speed TCP performs compared to NewReno during times of large bandwidths.

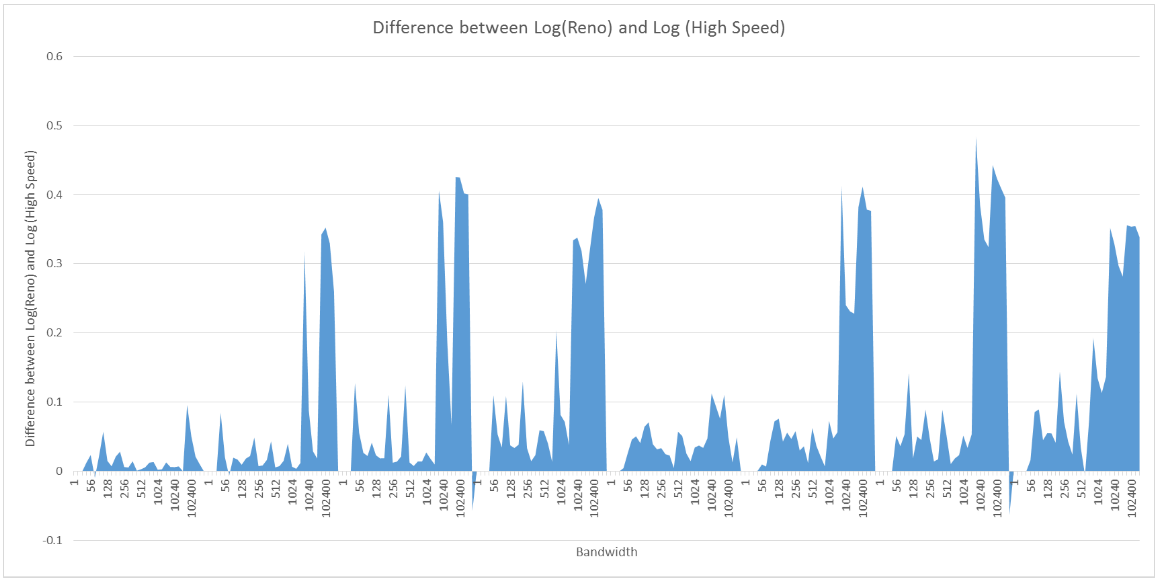
We can also see the impact of propagation time. When propagation time is low High Speed TCP and NewReno perform about the same. They always perform the same with low number of nodes and low propagation time because High Speed TCP has not fully re-written TCP, just optimized for larger throughput.



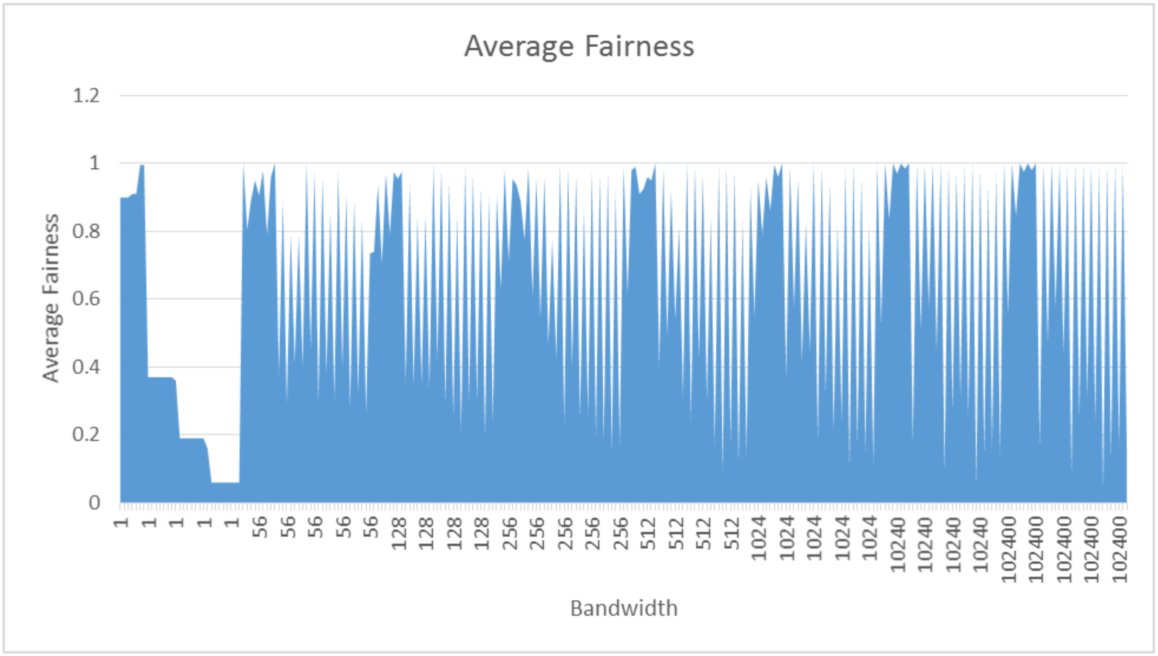
Below is a better look at the impact of propagation time. As propagation time increases and bandwidth increases it creates the best-case scenario for High Speed TCP. When the propagation time reaches one second we see the amount of data sent over a variety of experiments increase further proving High Speed TCP is built for long distances.



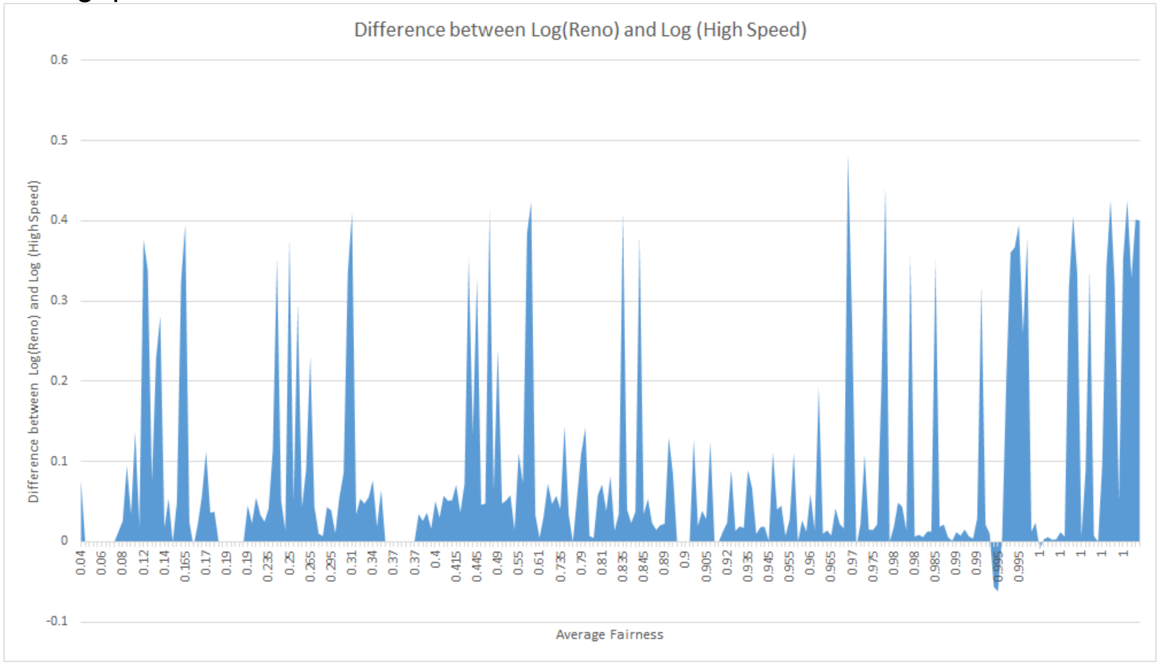
The graph below is divided by the delay factors with 0 on the left and 100 on the right. Both sides look symmetrical at first glance, and over all it is true delay doesn't play a large roll. However, High Speed TCP boots throughput in higher delay cases.



We also tracked the fairness each node had in during our simulations. The number of nodes increases from left to right, while propagation time increases from left to right and starts again faster than the number of nodes. We can clearly see the impact number of nodes has on fairness when the bandwidth is 1kbps. With each larger set of nodes the fairness drops. We can also see when there are only two nodes, the fairness is almost 1 in during every experiment.



Below the average fairness of the experiments is increasing from the left to the right. The averages were from the same experiment settings but different protocols. We can see there is no little correlation between fairness and throughput.



Conclusions

High Speed TCP produces more throughput than NewReno during very specific network environments. If a network has a low bandwidth, and low propagation time the two protocols will preform very similarly. However, in a network with high bandwidth, high propagation time High Speed TCP out preforms NewReno.

Testing gigabit speeds would be the next step of this study, however we did not have access to the resources to run such simulations.

Driver.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NB_SIMULATIONS 30
#define NB_BANDWIDTH_VALUES 8
#define NB_DELAY_TIMES 2
#define NB_PROPOGATION_TIMES 4
#define NB_NODE_VALUES 4

#define PROPAGATION_DEVIATION 12

void error(char *str);

void main (int argc, char **argv){
  FILE *fpthgpt, *output;
  double *thgpt;
  double *sumThgpt;
  double *hsThgpt;
  double *hsSumThgpt;
  double *sumFairnessReno;
  double *sumFairnessHighSpeed;


  long   BandwidthValues[NB_BANDWIDTH_VALUES] =
  {1, 56, 128, 256, 512, 1024, 10240, 102400}; /*in bps*/
//{1kbps, 56kbps, 128kbps, 256kbps, 512kbps, 1Mbps, 10Mbps, 100Mbps};

  int   PropagationTimes[NB_PROPOGATION_TIMES] =
  {50, 250, 500, 1000}; /* in milliseconds */

  int   DelayTimes[NB_DELAY_TIMES] =
  {0, 100}; /* in milliseconds */

  int   NodeValues[NB_NODE_VALUES] =
  {2, 8, 16, 32};

  int BANDWIDTH_COUNTER, PROPOGATION_COUNTER,
DELAY_COUNTER, NODE_COUNTER, EXPERIMENT_COUNTER=0;
  int i;
  char ns_command[512];

  srand(time(NULL));
  for(BANDWIDTH_COUNTER = 0; BANDWIDTH_COUNTER <
NB_BANDWIDTH_VALUES; BANDWIDTH_COUNTER++ ) {
    long bandwidthValue = BandwidthValues[BANDWIDTH_COUNTER]; // for
readability

    for(PROPOGATION_COUNTER = 0; PROPOGATION_COUNTER <
NB_PROPOGATION_TIMES; PROPOGATION_COUNTER++ ) {
      int propagationValue = PropagationTimes[PROPOGATION_COUNTER];

      for(DELAY_COUNTER = 0; DELAY_COUNTER < NB_DELAY_TIMES;
DELAY_COUNTER++ ) {
        int delayValue = DelayTimes[DELAY_COUNTER];

        for( NODE_COUNTER = 0; NODE_COUNTER < NB_NODE_VALUES;
NODE_COUNTER++ ) {
          int nodeValue = NodeValues[NODE_COUNTER];
            double renoFairness_bottom =0, highSpeedFairness_bottom =0,
renoSum=0, highSpeedSum =0, renoAverage =0, highSpeedAverage =0;

          thgpt = (double*)malloc( sizeof(double) * nodeValue);
          sumThgpt = (double*)malloc( sizeof(double) * nodeValue);
          hsThgpt = (double*)malloc( sizeof(double) * nodeValue);
          hsSumThgpt = (double*)malloc( sizeof(double) * nodeValue);
          sumFairnessReno = (double*)malloc( sizeof(double) * nodeValue);
          sumFairnessHighSpeed = (double*)malloc( sizeof(double) * nodeValue);

          for( i=0; i < nodeValue; i++ ) {
            thgpt[i] = 0;
            sumThgpt[i] = 0;
            hsThgpt[i] = 0;
            hsSumThgpt[i] = 0;
            sumFairnessReno[i] = 0;
            sumFairnessHighSpeed[i] =0;
          }

          printf("\n\n >>>> Experiment %d <<<<\n", ++EXPERIMENT_COUNTER);
          for( i=0; i<NB_SIMULATIONS; i++ ) {
              printf("--Sim:%d --\n",i);
            int randValue = rand() % (PROPAGATION_DEVIATION *
propagationValue);
            //Reno Run
            sprintf(ns_command,"ns reno.tcl %ld %d %d %d %d\n",
                    bandwidthValue, propagationValue, delayValue, nodeValue,
randValue);
            printf("%s", ns_command);
            system(ns_command);
            fpthgpt = fopen("thgt.tr","r");

            if (!fpthgpt)
              error("cannot open thgpt\n");

              printf("Node throughput: [");
            int j=0;
            while( fscanf(fpthgpt, "%lf", &thgpt[j]) == 1 ) {
              sumThgpt[j] += thgpt[j];
              printf("(%d = %6.2lf )", j, thgpt[j]);
              j++;
            }

              fclose(fpthgpt);

            printf("]\n");

            // High Speed Run
            sprintf(ns_command,"ns hstcp.tcl %ld %d %d %d %d\n",
                    bandwidthValue, propagationValue, delayValue, nodeValue,
randValue);
            printf("%s", ns_command);
            system(ns_command);

            fpthgpt = fopen("hsthgt.tr","r");

            if (!fpthgpt)
              error("cannot open hsthgpt\n");

              printf("Node Throughput: [");
            j=0;
            while( fscanf(fpthgpt, "%lf", &hsThgpt[j]) == 1 ) {
              hsSumThgpt[j] += hsThgpt[j];
              printf("(%d = %6.2lf )", j, hsThgpt[j]);
              j++;
            }

              fclose(fpthgpt);
              printf("]\n");
          }

          printf("\n >>>> Averages per node <<<<\n");
          for( i=0; i < nodeValue; i++ ) {
            printf("Avg Thgt%d = %6.2lf (Kbps)\n", i, sumThgpt[i] / 1000.0 /
NB_SIMULATIONS);
```

```
          printf("HS Avg Thgt%d = %6.2lf (Kbps)\n", i, hsSumThgpt[i] / 1000.0 /
NB_SIMULATIONS);
        }

          //calculates sum
        for( i=0; i < nodeValue; i++ ) {
          renoSum += sumThgpt[i]/ NB_SIMULATIONS;
          highSpeedSum += hsSumThgpt[i]/ NB_SIMULATIONS;
          renoFairness_bottom += (sumThgpt[i]/ NB_SIMULATIONS) *
(sumThgpt[i]/ NB_SIMULATIONS);
          highSpeedFairness_bottom += (hsSumThgpt[i]/ NB_SIMULATIONS) *
(hsSumThgpt[i]/ NB_SIMULATIONS);
        }

          renoAverage = renoSum / 1000.0  / nodeValue;
        highSpeedAverage = highSpeedSum / 1000.0  / nodeValue;

          //output info
          //printf("Fairness Sum: \n Reno: %6.2lf  HS: %6.2lf  \n Fariness Top:  \n
Reno: %6.2lf   HS: %6.2lf \n Fairness Bottom \n Reno: %6.2lf  HS:
%6.2lf",renoSum, highSpeedSum, renoSum * renoSum, highSpeedSum *
highSpeedSum,  nodeValue * renoFairness_bottom, nodeValue*
highSpeedFairness_bottom);

          printf("\n\n >>>>> Overall Stats <<<<< \n");
          printf("Average Throughput Reno: %6.2lf \n", renoAverage);
        printf("Average Throughput High Speed: %6.2lf \n", highSpeedAverage);
        printf("Fairness Reno: %6.2lf \n", (renoSum * renoSum) / (nodeValue *
renoFairness_bottom) );
        printf("Fairness High Speed: %6.2lf \n", (highSpeedSum * highSpeedSum)
/ (nodeValue * highSpeedFairness_bottom));

          // Write to the output file

          output = fopen("output.txt","a+");

          if (!output)
            error("Error Opening output file 'output.txt'\n");

          fprintf(output,"%d, %ld, %d, %d, %d,%6.2lf,%6.2lf,%6.2lf,%6.2lf, \n",
EXPERIMENT_COUNTER, bandwidthValue, propagationValue, delayValue,
nodeValue, renoSum, highSpeedSum,(renoSum * renoSum) / (nodeValue *
renoFairness_bottom), (highSpeedSum * highSpeedSum) / (nodeValue *
highSpeedFairness_bottom));
```

```
          fclose(output);
        free(thgpt);
        free(sumThgpt);
        free(hsThgpt);
          free(hsSumThgpt);
          free(sumFairnessReno);
          free(sumFairnessHighSpeed);
      }
    }
  }
 }
}



void error(char *str){
 printf("%s",str);
 exit(1);
}
```

hstcp.tcl

```
#Define a 'finish' procedure
proc finish {} {
        global ns nf hstcp filetrace StartTimeInterval N

        $ns flush-trace

  #Close out
        close $nf

    # Collecting Throughput
      set tf   [open hsthgt.tr w]
      set time [$ns now]

      set thgptResult ""

      for {set i 1} {$i <= $N} {incr i} {
        set thgpt($i) [$hstcp($i) set ack_]

        if { $thgpt($i) > 0  } {
          set thgpt($i) [expr $thgpt($i) * 1000.0 * 8.0 / ($time - ($StartTimeInterval /
$N * $i)) ]
        } else {
          set thgpt($i) 0.0
        }

        set thgptResult [concat $thgptResult $thgpt($i)]
      }

    # assumes packet of 1000 bytes
      #set thgpt0 [expr $thgpt0 * 1000.0 * 8.0 / ($time - 0.200) ]

      puts $tf $thgptResult
      close $tf

  #Execute nam on the trace file
        #exec nam out.nam &
        exit 0
}

# This program has 6 arguments
# - Bw [bandwith of the bottleneck]
# - Tp [propagation time]
# - D  [delay]
# - N  [number of senders and receivers]
```

```
if {$argc != 5} {
  puts "$argc : Incorrect number of arguments"
  puts "usage:  ns exp.tcl Bw(kbps) Tp(ms) D(ms) N  StartTimeInterval(ms)"
  exit 0
}

set Bw [lindex $argv  0]
append Bw Kb

set Tp [lindex $argv  1]
append Tp ms

set D  [lindex $argv  2]
set D [expr $D / 1000.0]

set N  [lindex $argv  3]

set StartTimeInterval [lindex $argv 4]
set StartTimeInterval [expr $StartTimeInterval / 1000.0];

puts "Bw = $Bw   Tp= $Tp    D = $D    N = $N  Start Time Interval =
$StartTimeInterval"

#Create a simulator object
set ns [new Simulator]

# Associated with network animator. Unnecessary for this project
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

# Code space

# ------- NODE CREATION ------- #
#Create N senders
for {set i 1} {$i <= $N} {incr i} {
     set s($i) [$ns node]
}

#Create N receivers
for {set i 1} {$i <= $N} {incr i} {
     set r($i) [$ns node]
}

#Create router nodes (Sending Router R, Destination Router Rd)
```

```
set R  [$ns node]
set Rd [$ns node]

# ------- LINK CREATION ------- #
set Step [expr (180 / $N) / 2]

#Create links between the sender nodes and sending router
for {set i 1} {$i <= $N} {incr i} {
      $ns duplex-link $s($i) $R $Bw [expr ($i -1)*($D)] DropTail
}

#Create links between the receiver nodes and destination router
for {set i 1} {$i <= $N} {incr i} {
      $ns duplex-link $r($i) $Rd $Bw [expr ($i -1)*($D)] DropTail
}

#Create link between routers
$ns duplex-link $R $Rd $Bw $Tp DropTail

# ------- HSTCP Connections ------- #
for {set i 1} {$i <= $N} {incr i} {
   set hstcp($i) [$ns create-connection TCP/Sack1 $s($i) TCPSink/Sack1 $r($i)
0]
   $hstcp($i) set window_ 10000
}

# ------- FTP Traffic ------- #
for {set i 1} {$i <= $N} {incr i} {
   set ftp($i) [$hstcp($i) attach-app FTP]
   $ns at [expr $StartTimeInterval / $N * $i] "$ftp($i) start"
}

#Call the finish procedure after 5 seconds simulation time
$ns at 50.0 "finish"

#Run the simulation
$ns run
```

reno.tcl

```
#Define a 'finish' procedure
proc finish {} {
        global ns nf tcp filetrace StartTimeInterval N

        $ns flush-trace

  #Close out
        close $nf

    # Collecting Throughput
      set tf   [open thgt.tr w]
      set time [$ns now]

      set thgptResult ""

      for {set i 1} {$i <= $N} {incr i} {
        set thgpt($i) [$tcp($i) set ack_]

        if { $thgpt($i) > 0  } {
          set thgpt($i) [expr $thgpt($i) * 1000.0 * 8.0 / ($time - ($StartTimeInterval /
$N * $i)) ]
        } else {
          set thgpt($i) 0.0
        }

        set thgptResult [concat $thgptResult $thgpt($i)]
      }

    # assumes packet of 1000 bytes
      #set thgpt0 [expr $thgpt0 * 1000.0 * 8.0 / ($time - 0.200) ]

      puts $tf $thgptResult
      close $tf

  #Execute nam on the trace file
        #exec nam out.nam &
        exit 0
}

# This program has 6 arguments
# - Bw [bandwith of the bottleneck]
# - Tp [propagation time]
# - D  [delay]
# - N  [number of senders and receivers]
```

```
if {$argc != 5} {
  puts "$argc : Incorrect number of arguments"
  puts "usage:  ns exp.tcl Bw(kbps) Tp(ms) D(ms) N StartTimeInterval(ms)"
  exit 0
}

set Bw [lindex $argv  0]
append Bw Kb

set Tp [lindex $argv  1]
append Tp ms

set D  [lindex $argv  2]
set D [expr $D / 1000.0]

set N  [lindex $argv  3]

set StartTimeInterval [lindex $argv 4]
set StartTimeInterval [expr $StartTimeInterval / 1000.0];

puts "Bw = $Bw   Tp= $Tp    D = $D    N = $N  Start Time Interval =
$StartTimeInterval"

#Create a simulator object
set ns [new Simulator]

# Associated with network animator. Unnecessary for this project
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

# Code space

# ------- NODE CREATION ------- #
#Create N senders
for {set i 1} {$i <= $N} {incr i} {
     set s($i) [$ns node]
}

#Create N receivers
for {set i 1} {$i <= $N} {incr i} {
     set r($i) [$ns node]
}

#Create router nodes (Sending Router R, Destination Router Rd)
```

```
set R  [$ns node]
set Rd [$ns node]

# ------- LINK CREATION ------- #
set Step [expr (180 / $N) / 2]

#Create links between the sender nodes and sending router
for {set i 1} {$i <= $N} {incr i} {
	$ns duplex-link $s($i) $R $Bw [expr ($i -1)*($D)] DropTail
}

#Create links between the receiver nodes and destination router
for {set i 1} {$i <= $N} {incr i} {
	$ns duplex-link $r($i) $Rd $Bw [expr ($i -1)*($D)] DropTail
}

#Create link between routers
$ns duplex-link $R $Rd $Bw $Tp DropTail

# ------- TCP Connections ------- #
for {set i 1} {$i <= $N} {incr i} {
   set tcp($i) [$ns create-connection TCP/Newreno $s($i) TCPSink $r($i) 0]
   $tcp($i) set window_ 10000
}

# ------- FTP Traffic ------- #
for {set i 1} {$i <= $N} {incr i} {
   set ftp($i) [$tcp($i) attach-app FTP]
   $ns at [expr $StartTimeInterval / $N * $i] "$ftp($i) start"
}

#Call the finish procedure after 5 seconds simulation time
$ns at 50.0 "finish"

#Run the simulation
$ns run
```

Raw Results

```
1, 1, 50, 0, 2,482.00,482.00,  0.90,  0.90,
2, 1, 50, 0, 8,480.74,480.74,  0.37,  0.37,
3, 1, 50, 0, 16,480.33,480.33,  0.19,  0.19,
4, 1, 50, 0, 32,320.09,320.09,  0.06,  0.06,
5, 1, 50, 100, 2,482.00,482.00,  0.90,  0.90,
6, 1, 50, 100, 8,480.71,480.71,  0.37,  0.37,
7, 1, 50, 100, 16,480.35,480.35,  0.19,  0.19,
8, 1, 50, 100, 32,320.09,320.09,  0.06,  0.06,
9, 1, 250, 0, 2,488.74,488.74,  0.90,  0.90,
10, 1, 250, 0, 8,483.34,483.34,  0.37,  0.37,
11, 1, 250, 0, 16,481.86,481.86,  0.19,  0.19,
12, 1, 250, 0, 32,320.41,320.41,  0.06,  0.06,
13, 1, 250, 100, 2,490.12,490.12,  0.90,  0.90,
14, 1, 250, 100, 8,483.26,483.26,  0.37,  0.37,
15, 1, 250, 100, 16,481.79,481.79,  0.19,  0.19,
16, 1, 250, 100, 32,272.39,272.39,  0.06,  0.06,
17, 1, 500, 0, 2,499.80,499.80,  0.91,  0.91,
18, 1, 500, 0, 8,488.15,488.15,  0.37,  0.37,
19, 1, 500, 0, 16,484.22,484.22,  0.19,  0.19,
20, 1, 500, 0, 32,267.51,267.51,  0.06,  0.06,
21, 1, 500, 100, 2,495.57,495.57,  0.91,  0.91,
22, 1, 500, 100, 8,487.65,487.65,  0.37,  0.37,
23, 1, 500, 100, 16,483.57,483.57,  0.19,  0.19,
24, 1, 500, 100, 32,299.58,299.58,  0.06,  0.06,
25, 1, 1000, 0, 2,400.70,351.94,  0.99,  1.00,
26, 1, 1000, 0, 8,478.20,478.20,  0.37,  0.37,
27, 1, 1000, 0, 16,481.53,481.53,  0.19,  0.19,
28, 1, 1000, 0, 32,257.45,257.45,  0.06,  0.06,
29, 1, 1000, 100, 2,405.41,351.37,  0.99,  1.00,
30, 1, 1000, 100, 8,426.14,426.14,  0.36,  0.36,
31, 1, 1000, 100, 16,367.70,367.70,  0.16,  0.16,
32, 1, 1000, 100, 32,257.64,257.64,  0.06,  0.06,
33, 56, 50, 0, 2,49806.34,51281.30,  1.00,  1.00,
34, 56, 50, 0, 8,40876.74,43165.03,  1.00,  1.00,
35, 56, 50, 0, 16,39314.29,38503.42,  1.00,  1.00,
36, 56, 50, 0, 32,38464.88,40471.66,  0.97,  0.98,
37, 56, 50, 100, 2,36747.33,37148.18,  0.81,  0.80,
38, 56, 50, 100, 8,41919.13,44504.17,  0.37,  0.39,
39, 56, 50, 100, 16,40392.41,44913.89,  0.58,  0.33,
40, 56, 50, 100, 32,35391.78,39772.73,  0.46,  0.34,
41, 56, 250, 0, 2,36574.60,44446.02,  0.85,  0.94,
42, 56, 250, 0, 8,44110.32,46269.69,  0.90,  0.88,
43, 56, 250, 0, 16,41663.01,41033.26,  0.97,  0.98,
44, 56, 250, 0, 32,38808.91,40620.39,  0.89,  0.92,
45, 56, 250, 100, 2,43715.76,43822.46,  0.95,  0.95,
46, 56, 250, 100, 8,45438.29,46507.74,  0.29,  0.29,
47, 56, 250, 100, 16,42897.20,43591.93,  0.30,  0.29,
48, 56, 250, 100, 32,35474.41,39228.77,  0.33,  0.23,
49, 56, 500, 0, 2,34272.77,45977.97,  0.95,  0.86,
50, 56, 500, 0, 8,40981.79,46568.12,  0.85,  0.73,
51, 56, 500, 0, 16,42813.70,45537.96,  0.95,  0.97,
52, 56, 500, 0, 32,40337.57,42463.15,  0.88,  0.90,
53, 56, 500, 100, 2,46837.58,46837.58,  0.98,  0.98,
54, 56, 500, 100, 8,41073.34,46207.84,  0.39,  0.43,
55, 56, 500, 100, 16,38192.84,41523.46,  0.41,  0.35,
56, 56, 500, 100, 32,35545.07,40221.16,  0.37,  0.27,
57, 56, 1000, 0, 2,43178.78,43178.78,  0.79,  0.79,
58, 56, 1000, 0, 8,34570.93,44528.81,  0.86,  0.72,
59, 56, 1000, 0, 16,39041.02,44162.87,  0.88,  0.83,
60, 56, 1000, 0, 32,40477.70,43822.50,  0.82,  0.85,
61, 56, 1000, 100, 2,42143.45,42143.45,  0.96,  0.96,
62, 56, 1000, 100, 8,36503.43,37892.99,  0.41,  0.39,
63, 56, 1000, 100, 16,30838.66,37577.42,  0.35,  0.26,
64, 56, 1000, 100, 32,30525.88,37517.10,  0.34,  0.19,
65, 128, 50, 0, 2,103474.85,118099.07,  0.77,  0.70,
66, 128, 50, 0, 8,110880.65,114831.38,  0.99,  0.96,
67, 128, 50, 0, 16,106958.03,108802.46,  1.00,  0.99,
68, 128, 50, 0, 32,103414.55,108658.28,  0.99,  0.99,
69, 128, 50, 100, 2,102561.90,112694.89,  0.76,  0.72,
70, 128, 50, 100, 8,78115.24,90628.21,  0.36,  0.35,
71, 128, 50, 100, 16,93842.51,110487.90,  0.49,  0.35,
72, 128, 50, 100, 32,93312.05,102150.32,  0.30,  0.30,
73, 128, 250, 0, 2,109685.01,114122.62,  0.98,  0.89,
74, 128, 250, 0, 8,111309.37,113805.98,  0.95,  0.93,
75, 128, 250, 0, 16,109099.90,113856.80,  0.98,  0.98,
76, 128, 250, 0, 32,103351.52,108789.40,  0.97,  0.97,
77, 128, 250, 100, 2,92619.73,109406.51,  0.74,  0.67,
78, 128, 250, 100, 8,78802.34,93922.55,  0.36,  0.33,
79, 128, 250, 100, 16,97391.69,107551.31,  0.29,  0.30,
80, 128, 250, 100, 32,88411.17,100573.72,  0.32,  0.28,
81, 128, 500, 0, 2,84035.70,92460.72,  0.99,  0.95,
82, 128, 500, 0, 8,108006.21,113989.61,  0.87,  0.81,
83, 128, 500, 0, 16,105820.97,110512.60,  0.95,  0.94,
84, 128, 500, 0, 32,103697.98,108364.79,  0.93,  0.93,
85, 128, 500, 100, 2,70410.07,97608.22,  0.88,  0.70,
86, 128, 500, 100, 8,92316.14,96370.92,  0.35,  0.35,
87, 128, 500, 100, 16,91366.60,102607.11,  0.29,  0.22,
88, 128, 500, 100, 32,89955.21,99854.39,  0.24,  0.15,
89, 128, 1000, 0, 2,63794.66,81987.85,  0.97,  0.98,
90, 128, 1000, 0, 8,94413.87,102967.97,  0.90,  0.79,
91, 128, 1000, 0, 16,101009.89,109199.13,  0.88,  0.82,
92, 128, 1000, 0, 32,97872.04,106954.60,  0.91,  0.90,
93, 128, 1000, 100, 2,67544.61,74937.70,  0.97,  0.94,
94, 128, 1000, 100, 8,70363.13,79907.81,  0.36,  0.32,
95, 128, 1000, 100, 16,78181.92,88744.08,  0.23,  0.19,
96, 128, 1000, 100, 32,79778.33,87759.42,  0.26,  0.22,
97, 256, 50, 0, 2,225471.57,240759.85,  0.87,  0.94,
98, 256, 50, 0, 8,234245.25,237686.55,  0.99,  0.98,
99, 256, 50, 0, 16,229534.97,232551.43,  0.99,  0.99,
100, 256, 50, 0, 32,223002.81,230636.76,  0.98,  0.97,
101, 256, 50, 100, 2,226451.60,243713.13,  0.71,  0.55,
102, 256, 50, 100, 8,185162.37,200070.06,  0.64,  0.58,
103, 256, 50, 100, 16,211145.48,223617.39,  0.23,  0.24,
104, 256, 50, 100, 32,201152.88,212098.63,  0.20,  0.20,
105, 256, 250, 0, 2,194383.22,217505.08,  0.96,  1.00,
106, 256, 250, 0, 8,232260.22,236320.66,  0.97,  0.94,
107, 256, 250, 0, 16,229573.59,234223.17,  0.99,  0.98,
108, 256, 250, 0, 32,223116.01,231804.19,  0.97,  0.97,
109, 256, 250, 100, 2,207747.61,231503.72,  0.78,  0.64,
110, 256, 250, 100, 8,164003.50,187481.26,  0.60,  0.49,
111, 256, 250, 100, 16,200027.67,214428.64,  0.41,  0.39,
112, 256, 250, 100, 32,194140.02,211062.35,  0.20,  0.16,
113, 256, 500, 0, 2,166362.91,214593.03,  0.98,  0.93,
114, 256, 500, 0, 8,222639.53,229188.80,  0.98,  0.94,
115, 256, 500, 0, 16,225754.37,233191.38,  0.97,  0.96,
116, 256, 500, 0, 32,219487.61,230662.89,  0.98,  0.97,
117, 256, 500, 100, 2,163339.64,200571.27,  1.00,  0.87,
118, 256, 500, 100, 8,173914.66,194047.68,  0.49,  0.44,
119, 256, 500, 100, 16,189938.21,196329.84,  0.28,  0.22,
120, 256, 500, 100, 32,188060.71,195856.22,  0.16,  0.14,
121, 256, 1000, 0, 2,117965.04,159035.47,  0.98,  0.80,
122, 256, 1000, 0, 8,186190.44,201240.63,  0.82,  0.73,
123, 256, 1000, 0, 16,200502.48,207450.41,  0.89,  0.84,
124, 256, 1000, 0, 32,208229.92,219795.53,  0.93,  0.91,
125, 256, 1000, 100, 2,92522.16,128928.69,  0.85,  0.70,
126, 256, 1000, 100, 8,144443.93,170444.55,  0.46,  0.39,
127, 256, 1000, 100, 16,145610.90,160706.85,  0.27,  0.25,
128, 256, 1000, 100, 32,146188.97,154580.48,  0.19,  0.15,
129, 512, 50, 0, 2,483367.37,484501.95,  1.00,  0.98,
130, 512, 50, 0, 8,482069.46,485532.97,  1.00,  1.00,
131, 512, 50, 0, 16,474449.78,480906.44,  1.00,  1.00,
132, 512, 50, 0, 32,461419.27,475051.01,  0.98,  0.99,
133, 512, 50, 100, 2,467398.49,472647.72,  0.62,  0.61,
134, 512, 50, 100, 8,411385.46,469508.68,  0.49,  0.31,
135, 512, 50, 100, 16,404683.84,455335.27,  0.28,  0.21,
136, 512, 50, 100, 32,440291.93,467283.85,  0.10,  0.08,
137, 512, 250, 0, 2,416102.94,459953.17,  1.00,  0.96,
138, 512, 250, 0, 8,468441.68,474684.51,  0.99,  0.98,
139, 512, 250, 0, 16,471106.99,479462.43,  0.99,  0.99,
140, 512, 250, 0, 32,458925.82,475389.12,  0.99,  0.99,
141, 512, 250, 100, 2,414591.50,426302.10,  1.00,  0.98,
142, 512, 250, 100, 8,375406.38,433881.78,  0.55,  0.43,
143, 512, 250, 100, 16,375962.97,408980.22,  0.42,  0.43,
144, 512, 250, 100, 32,393165.53,412939.27,  0.17,  0.17,
145, 512, 500, 0, 2,299861.60,398669.87,  0.98,  0.84,
146, 512, 500, 0, 8,435466.50,448630.93,  0.95,  0.89,
147, 512, 500, 0, 16,450966.44,459198.11,  0.97,  0.96,
148, 512, 500, 0, 32,453323.76,468698.08,  0.98,  0.98,
149, 512, 500, 100, 2,302185.34,370893.68,  0.98,  0.87,
150, 512, 500, 100, 8,328020.73,369301.30,  0.52,  0.56,
151, 512, 500, 100, 16,352075.47,360865.44,  0.30,  0.30,
152, 512, 500, 100, 32,355969.90,371729.98,  0.12,  0.12,
153, 512, 1000, 0, 2,182089.25,208830.68,  0.94,  0.98,
154, 512, 1000, 0, 8,322400.14,368589.96,  0.84,  0.78,
155, 512, 1000, 0, 16,358309.48,393080.83,  0.86,  0.82,
156, 512, 1000, 0, 32,406652.19,419849.34,  0.88,  0.78,
157, 512, 1000, 100, 2,181488.26,234905.76,  0.92,  0.98,
158, 512, 1000, 100, 8,196261.49,212389.01,  0.31,  0.32,
159, 512, 1000, 100, 16,214576.00,211892.37,  0.16,  0.18,
160, 512, 1000, 100, 32,235812.00,280083.26,  0.14,  0.14,
161, 1024, 50, 0, 2,936682.13,966349.70,  0.96,  0.89,
162, 1024, 50, 0, 8,971624.33,976824.48,  1.00,  1.00,
163, 1024, 50, 0, 16,965549.17,972432.97,  1.00,  1.00,
164, 1024, 50, 0, 32,936321.28,964836.72,  0.98,  0.99,
165, 1024, 50, 100, 2,937794.51,970303.95,  0.57,  0.53,
166, 1024, 50, 100, 8,847693.05,918005.52,  0.35,  0.39,
167, 1024, 50, 100, 16,838118.91,913482.44,  0.18,  0.18,
168, 1024, 50, 100, 32,848799.97,917981.23,  0.10,  0.11,
169, 1024, 250, 0, 2,822782.99,901845.43,  1.00,  0.90,
170, 1024, 250, 0, 8,929403.53,944120.24,  1.00,  0.98,
171, 1024, 250, 0, 16,945092.12,953335.79,  0.99,  0.99,
172, 1024, 250, 0, 32,934054.51,960472.84,  1.00,  1.00,
173, 1024, 250, 100, 2,828905.74,842883.23,  0.77,  0.81,
174, 1024, 250, 100, 8,577691.35,683373.06,  0.59,  0.56,
175, 1024, 250, 100, 16,616229.38,687576.14,  0.32,  0.32,
176, 1024, 250, 100, 32,626597.85,713917.48,  0.17,  0.17,
177, 1024, 500, 0, 2,617287.87,638261.85,  0.94,  0.98,
178, 1024, 500, 0, 8,803266.82,855567.67,  0.97,  0.94,
179, 1024, 500, 0, 16,849637.64,886076.12,  0.94,  0.94,
180, 1024, 500, 0, 32,894060.52,914837.54,  0.97,  0.95,
181, 1024, 500, 100, 2,564427.54,595884.54,  0.82,  0.90,
182, 1024, 500, 100, 8,548440.36,617804.48,  0.38,  0.45,
183, 1024, 500, 100, 16,587093.36,634888.30,  0.19,  0.24,
184, 1024, 500, 100, 32,555722.02,628839.46,  0.14,  0.16,
185, 1024, 1000, 0, 2,224728.81,358933.56,  0.99,  1.00,
186, 1024, 1000, 0, 8,492713.53,594652.35,  0.85,  0.80,
187, 1024, 1000, 0, 16,573444.76,676106.90,  0.84,  0.80,
188, 1024, 1000, 0, 32,713286.98,778905.56,  0.84,  0.80,
189, 1024, 1000, 100, 2,192188.46,299432.54,  0.95,  0.97,
190, 1024, 1000, 100, 8,289841.78,395333.21,  0.45,  0.44,
191, 1024, 1000, 100, 16,300350.80,389889.20,  0.24,  0.24,
192, 1024, 1000, 100, 32,317529.09,434908.92,  0.12,  0.11,
193, 10240, 50, 0, 2,8894900.62,9027997.29,  1.00,  1.00,
194, 10240, 50, 0, 8,9440293.90,9575862.82,  1.00,  1.00,
195, 10240, 50, 0, 16,9351678.48,9508863.42,  1.00,  1.00,
196, 10240, 50, 0, 32,9618023.50,9623761.90,  1.00,  1.00,
197, 10240, 50, 100, 2,8319600.48,9284220.90,  0.53,  0.52,
198, 10240, 50, 100, 8,5731199.76,7425818.99,  0.18,  0.17,
199, 10240, 50, 100, 16,5862742.12,7289002.40,  0.09,  0.09,
200, 10240, 50, 100, 32,5952954.17,7096322.82,  0.05,  0.05,
201, 10240, 250, 0, 2,1468551.72,3046340.96,  1.00,  1.00,
202, 10240, 250, 0, 8,6330532.81,7750252.46,  1.00,  1.00,
203, 10240, 250, 0, 16,7547123.97,8068238.49,  0.99,  0.99,
204, 10240, 250, 0, 32,7983809.02,8333091.91,  0.99,  0.98,
205, 10240, 250, 100, 2,903660.74,2332444.41,  0.82,  0.85,
206, 10240, 250, 100, 8,1482611.24,2578352.43,  0.50,  0.52,
207, 10240, 250, 100, 16,1542538.94,2627865.03,  0.26,  0.28,
208, 10240, 250, 100, 32,1525168.10,2578531.05,  0.14,  0.14,
209, 10240, 500, 0, 2,434181.32,1106297.78,  1.00,  1.00,
210, 10240, 500, 0, 8,2005090.51,4598483.11,  1.00,  0.99,
211, 10240, 500, 0, 16,4273789.36,6579888.86,  0.99,  0.96,
212, 10240, 500, 0, 32,5973282.98,6965797.20,  0.97,  0.90,
213, 10240, 500, 100, 2,353247.53,1076780.11,  0.96,  0.98,
214, 10240, 500, 100, 8,699907.79,1695585.27,  0.61,  0.54,
215, 10240, 500, 100, 16,769943.68,1666280.75,  0.33,  0.29,
216, 10240, 500, 100, 32,801337.62,1690234.68,  0.18,  0.15,
217, 10240, 1000, 0, 2,211897.32,457265.03,  1.00,  1.00,
218, 10240, 1000, 0, 8,863962.20,1881259.54,  1.00,  1.00,
219, 10240, 1000, 0, 16,1634699.61,3405243.29,  0.99,  0.99,
220, 10240, 1000, 0, 32,3051338.10,5693352.11,  0.98,  0.96,
221, 10240, 1000, 100, 2,187869.41,422279.70,  0.99,  0.98,
222, 10240, 1000, 100, 8,302996.63,646133.39,  0.49,  0.41,
223, 10240, 1000, 100, 16,328248.87,650909.98,  0.29,  0.22,
224, 10240, 1000, 100, 32,351058.74,671308.93,  0.16,  0.12,
225, 102400, 50, 0, 2,37290663.24,46521448.83,  1.00,  1.00,
226, 102400, 50, 0, 8,69346838.88,77923019.92,  1.00,  1.00,
227, 102400, 50, 0, 16,79142361.46,83112817.20,  0.99,  0.99,
228, 102400, 50, 0, 32,82683513.50,84664112.33,  0.99,  0.99,
229, 102400, 50, 100, 2,19343131.10,24948998.40,  0.54,  0.57,
230, 102400, 50, 100, 8,18246705.48,20446846.05,  0.15,  0.17,
231, 102400, 50, 100, 16,18023033.05,18560045.43,  0.07,  0.09,
232, 102400, 50, 100, 32,17419151.72,19510078.37,  0.04,  0.04,
233, 102400, 250, 0, 2,1370550.27,3015134.92,  1.00,  1.00,
234, 102400, 250, 0, 8,5402848.33,12162242.56,  1.00,  1.00,
235, 102400, 250, 0, 16,11963256.54,25553352.23,  1.00,  1.00,
236, 102400, 250, 0, 32,25653731.35,46660815.89,  1.00,  0.99,
237, 102400, 250, 100, 2,904004.34,2178238.99,  0.81,  0.88,
238, 102400, 250, 100, 8,1446643.09,3732696.03,  0.45,  0.49,
239, 102400, 250, 100, 16,1507780.81,3603186.52,  0.24,  0.26,
240, 102400, 250, 100, 32,1519619.60,3617432.33,  0.13,  0.13,
241, 102400, 500, 0, 2,409882.32,1091328.23,  1.00,  1.00,
242, 102400, 500, 0, 8,1674500.03,4454535.70,  1.00,  1.00,
243, 102400, 500, 0, 16,3405679.39,8591679.85,  1.00,  1.00,
244, 102400, 500, 0, 32,7007816.64,17620956.78,  1.00,  1.00,
245, 102400, 500, 100, 2,342230.26,948986.00,  0.97,  0.98,
246, 102400, 500, 100, 8,669920.70,1777237.41,  0.62,  0.53,
247, 102400, 500, 100, 16,712372.95,1828366.55,  0.34,  0.28,
248, 102400, 500, 100, 32,735266.20,1829221.51,  0.19,  0.14,
249, 102400, 1000, 0, 2,217436.82,454907.18,  1.00,  1.00,
250, 102400, 1000, 0, 8,834160.34,1942645.51,  1.00,  0.99,
251, 102400, 1000, 0, 16,1617513.01,4018727.10,  1.00,  0.99,
252, 102400, 1000, 0, 32,3369057.43,8042797.80,  1.00,  0.99,
253, 102400, 1000, 100, 2,183901.48,417315.97,  0.98,  0.98,
254, 102400, 1000, 100, 8,299686.25,676683.07,  0.49,  0.39,
255, 102400, 1000, 100, 16,326712.37,739096.60,  0.28,  0.20,
256, 102400, 1000, 100, 32,343827.61,749576.43,  0.16,  0.10,
```