NS-2/tcptrace Project Report

ECE 4607

1 May, 2015

Enmao Diao

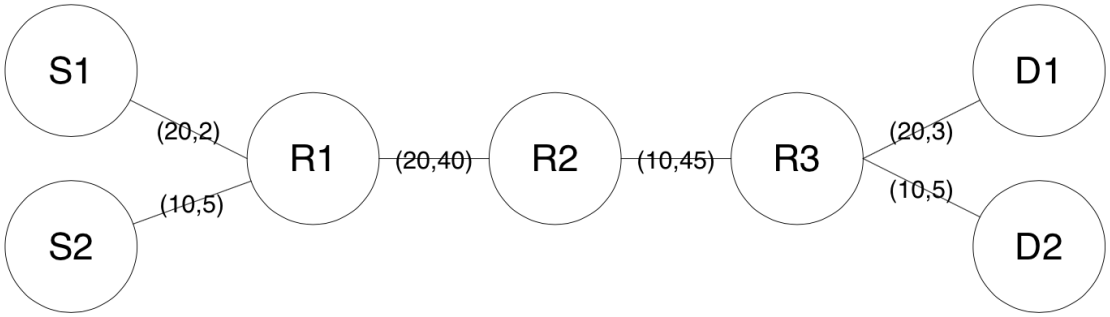
James Eccles

Daniel Graves

Wes Smith

Section 1 - Tcl script located in Appendix A

1. The following table shows the number of TCP segments received at the destination and the overall throughput for each flow. Simulations performed for TCP New Reno, Reno, SACK, and Tahoe.



Network Topology

In our simulations, there are four flows: Flow 1 is S1 - D1, Flow 2 is S1 - D2, Flow 3 is S2 - D1, Flow 4 is S2 - D2.

Table I

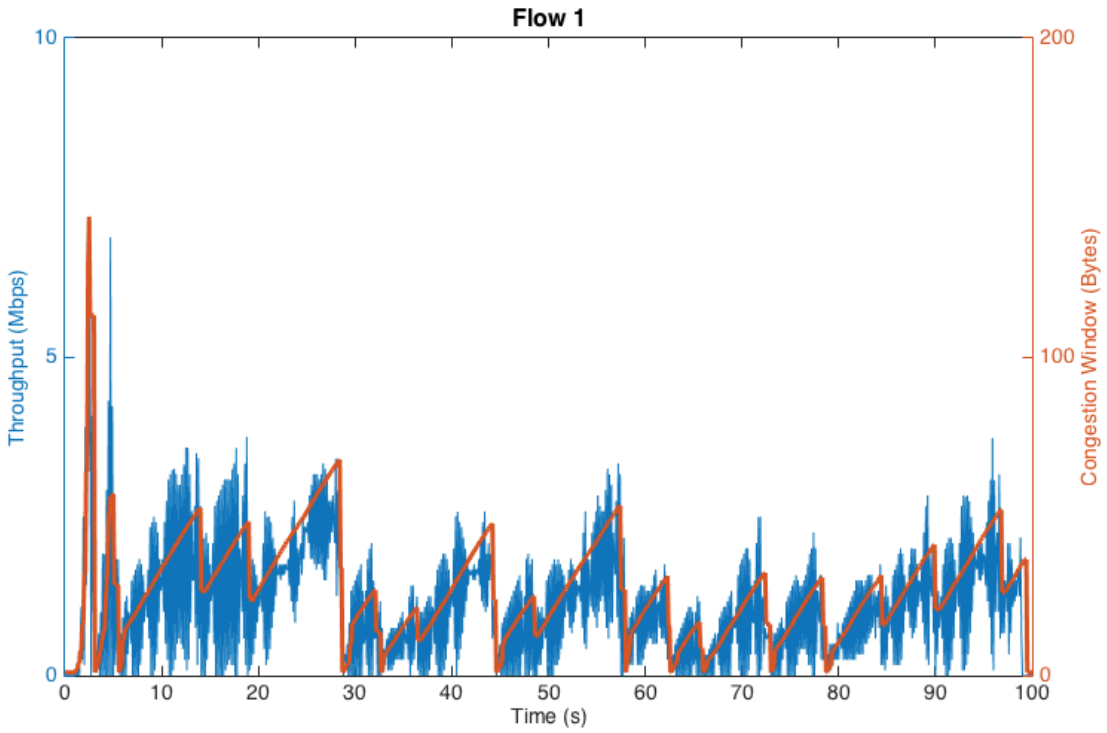
TCP Segments and Overall Throughput

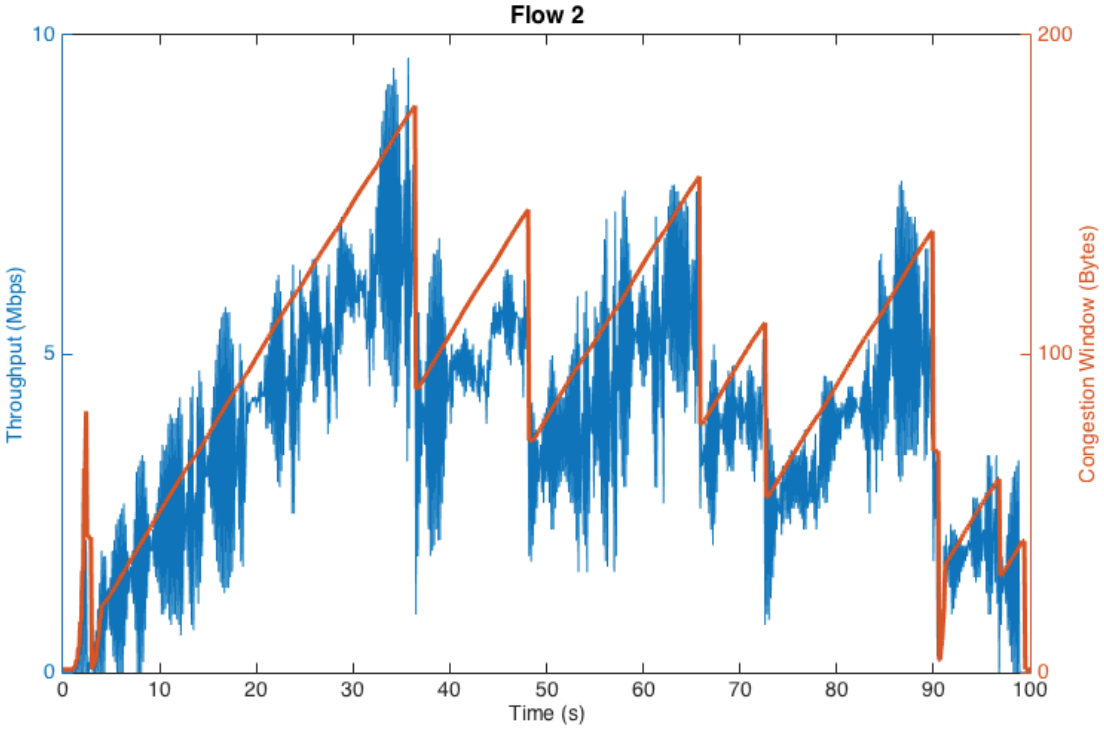
*Throughput in Mbps		**Flow**			
TCP Variant		1	2	3	4
New Reno	Segments	13351	45852	19096	27839
	Throughput	1.1312	3.8850	1.6179	2.3588
Reno	Segments	14625	38180	19544	30046
	Throughput	1.2390	3.2347	1.6558	2.5456
SACK	Segments	14477	38759	20207	27851

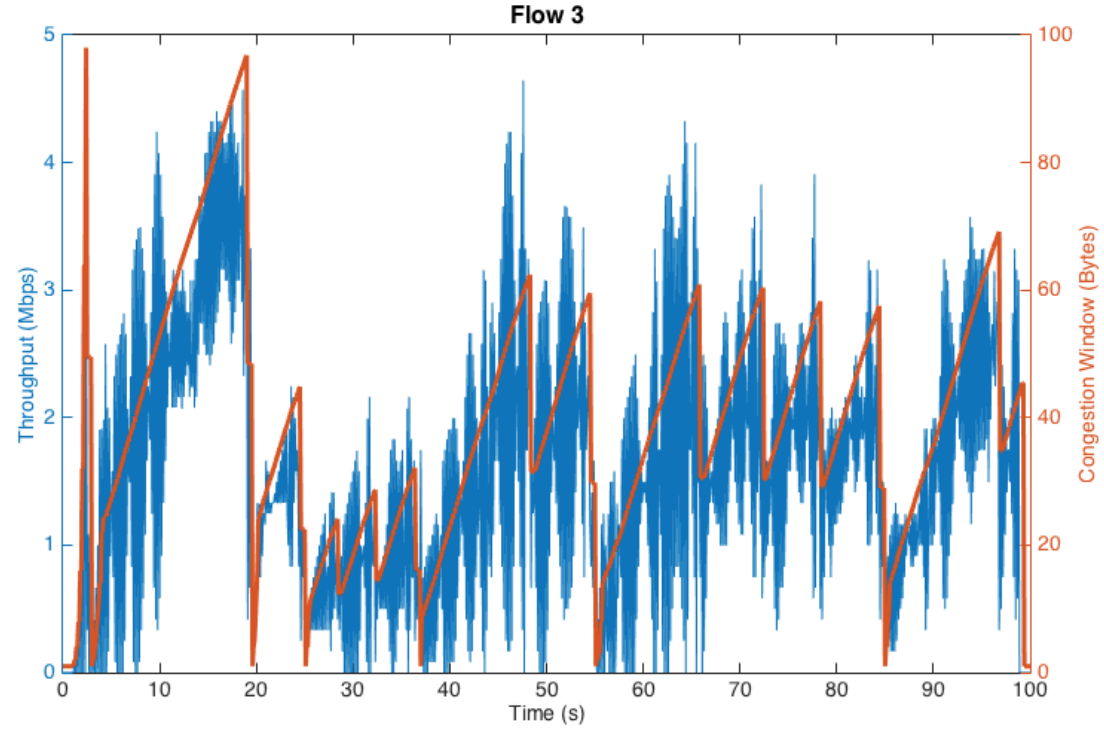
	Throughput	1.2266	3.2840	1.7121	2.3597
Tahoe	Segments	14477	38759	20207	27851
	Throughput	1.2266	3.2840	1.7121	2.3597

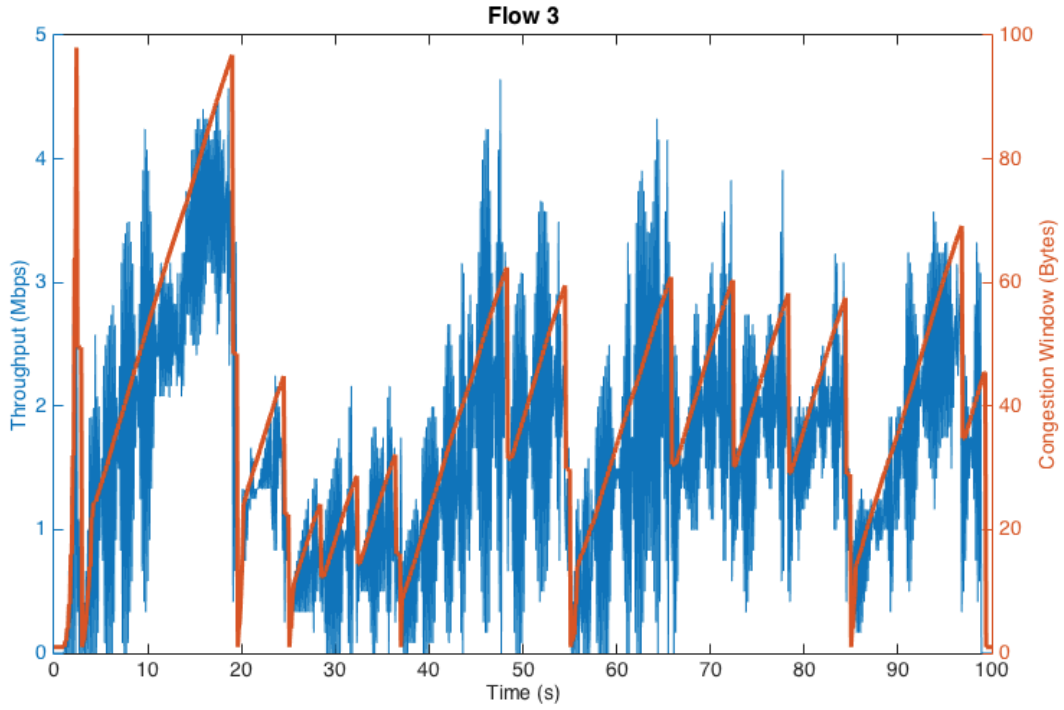
2. Plots of throughput and congestion window vs time of each flow:

New Reno

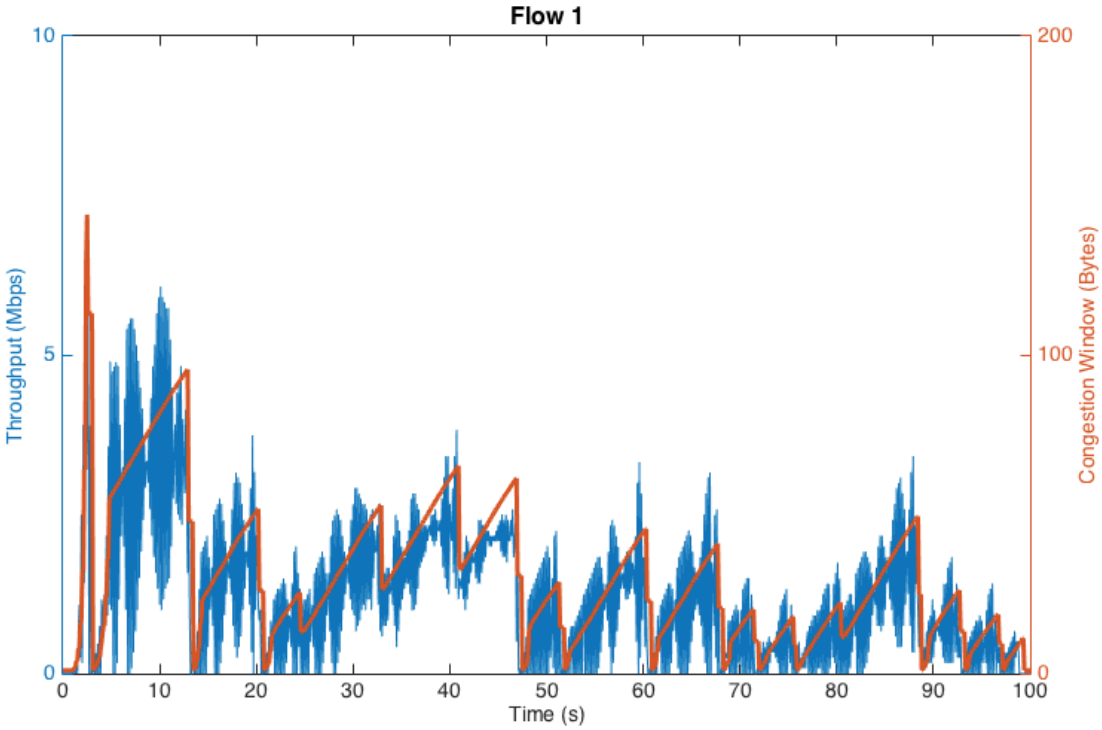


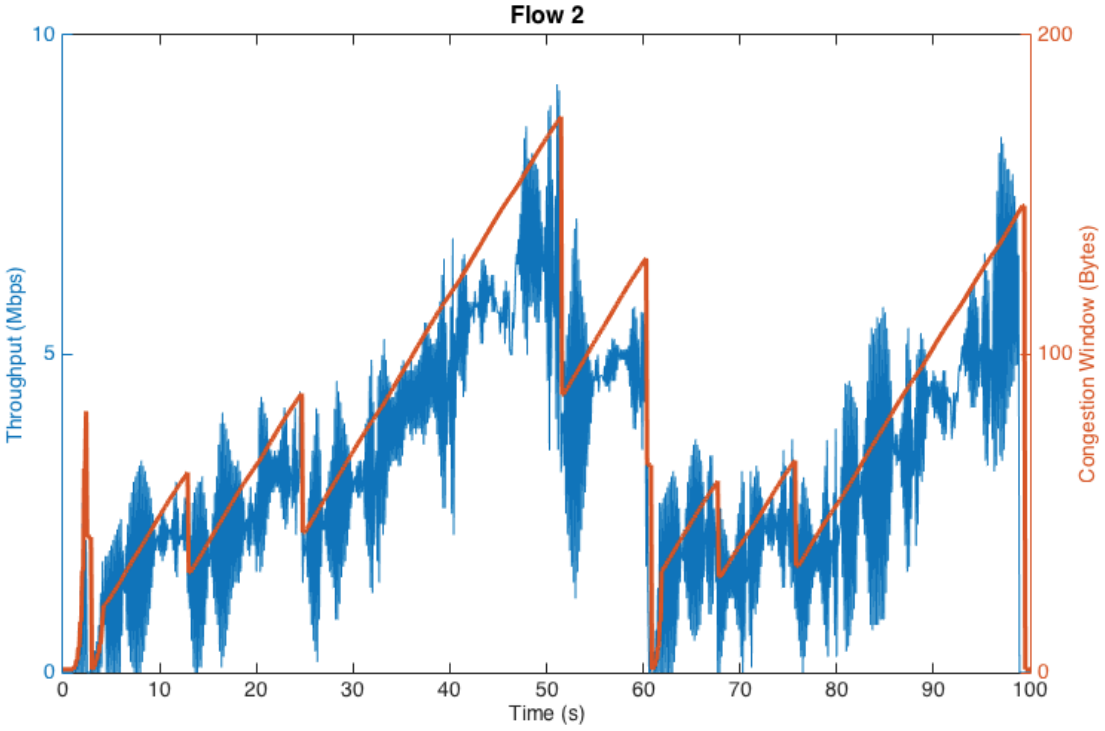

Flow 2
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
5
10
0
100
200
0
10
20
30
40
50
60
70
80
90
100

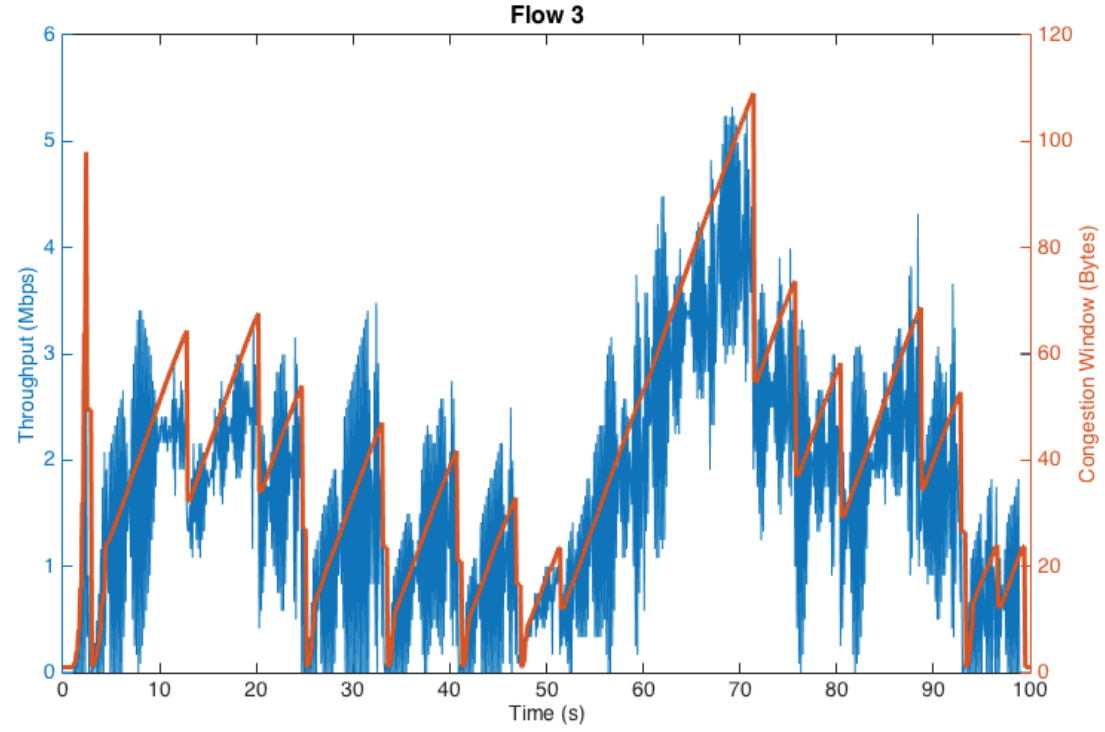

Flow 3
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
1
2
3
4
5
0
20
40
60
80
100
0
10
20
30
40
50
60
70
80
90
100

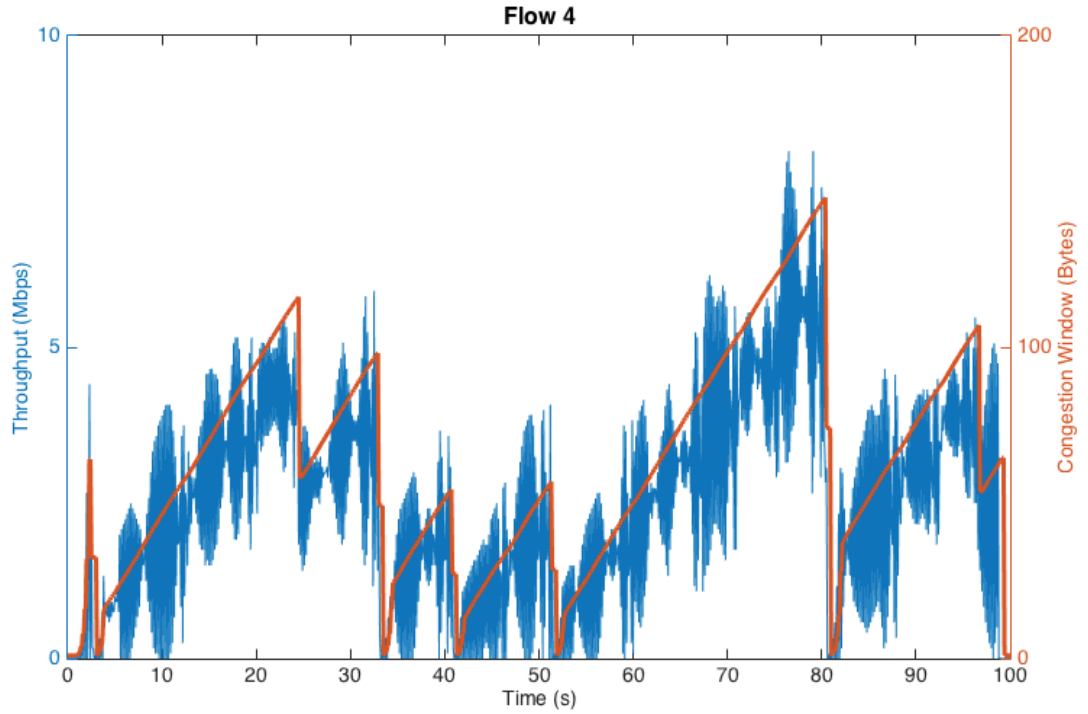

Flow 3
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
1
2
3
4
5
0
20
40
60
80
100
0
10
20
30
40
50
60
70
80
90
100

Reno

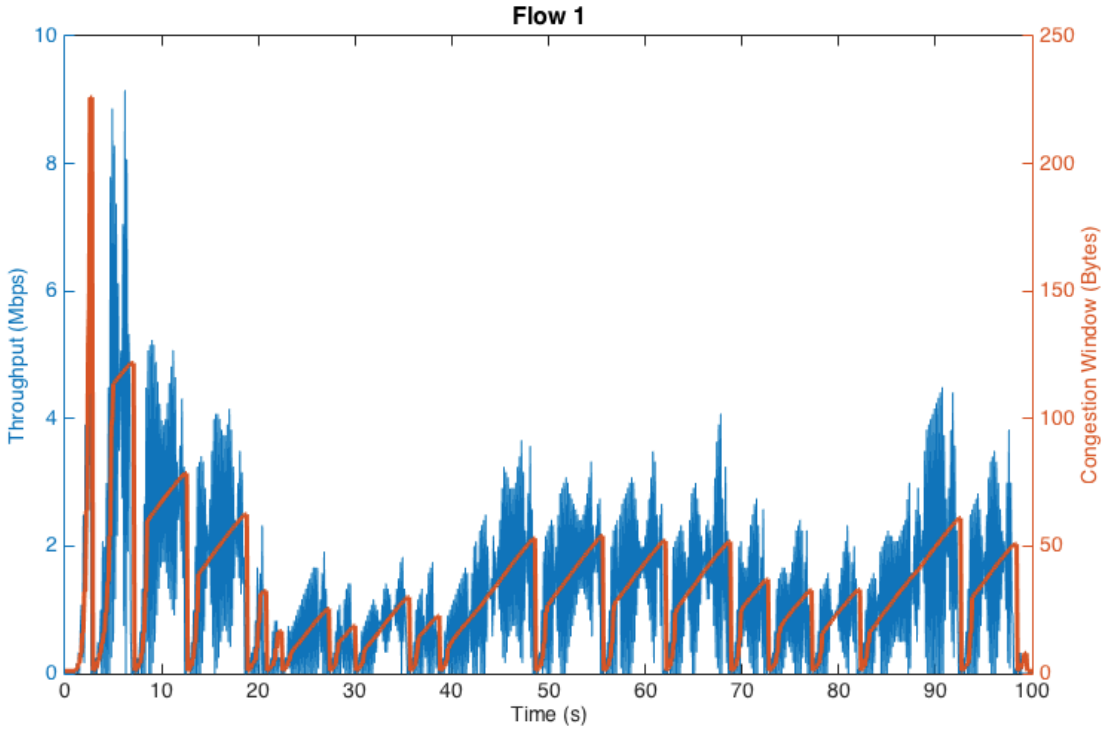

Flow 1
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
5
10
0
100
200
0
10
20
30
40
50
60
70
80
90
100

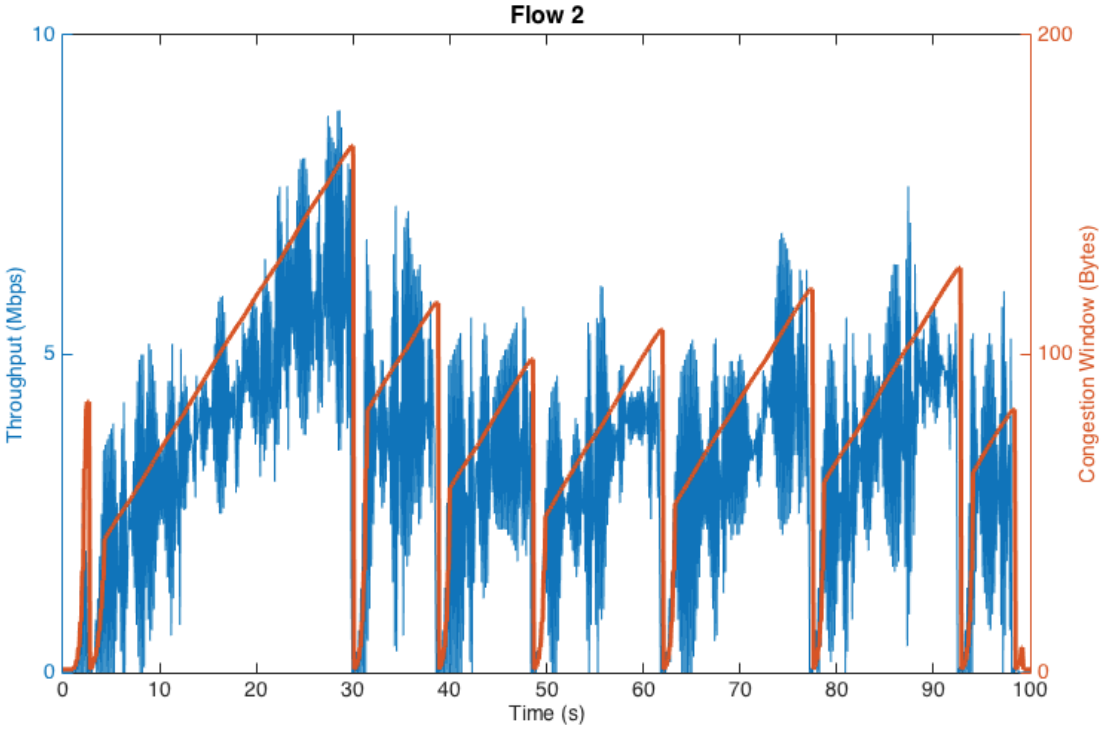

Flow 2
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
5
10
0
100
200
0
10
20
30
40
50
60
70
80
90
100

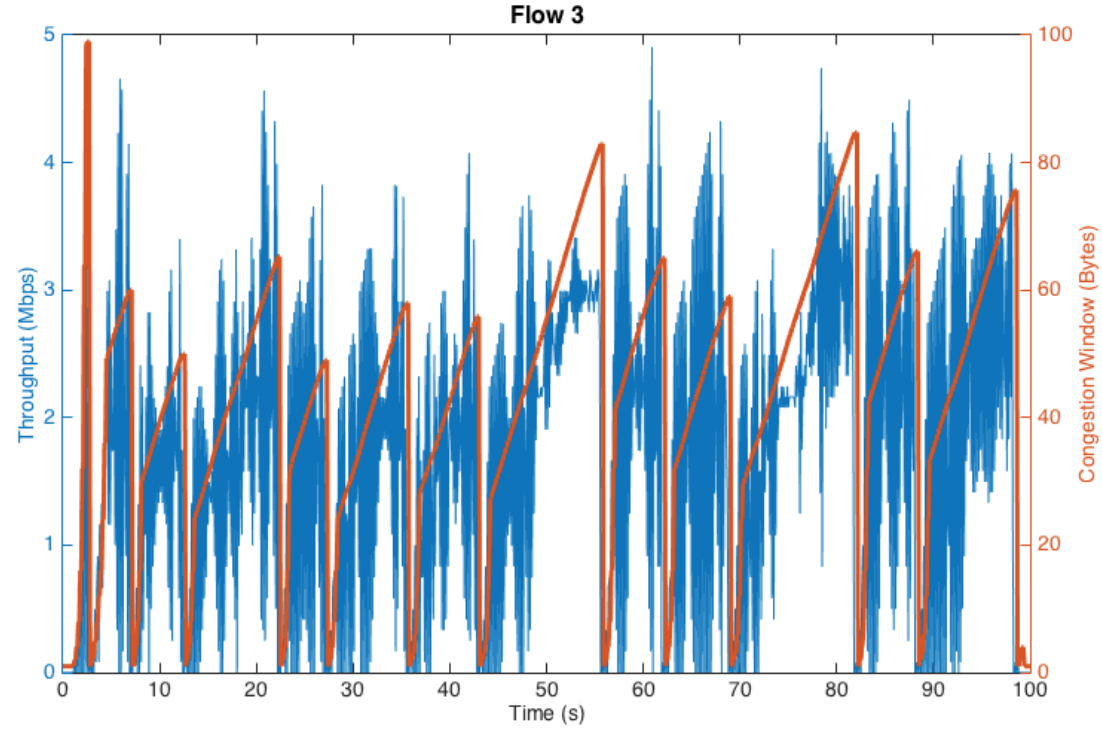

Flow 3
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
1
2
3
4
5
6
0
20
40
60
80
100
120
0
10
20
30
40
50
60
70
80
90
100

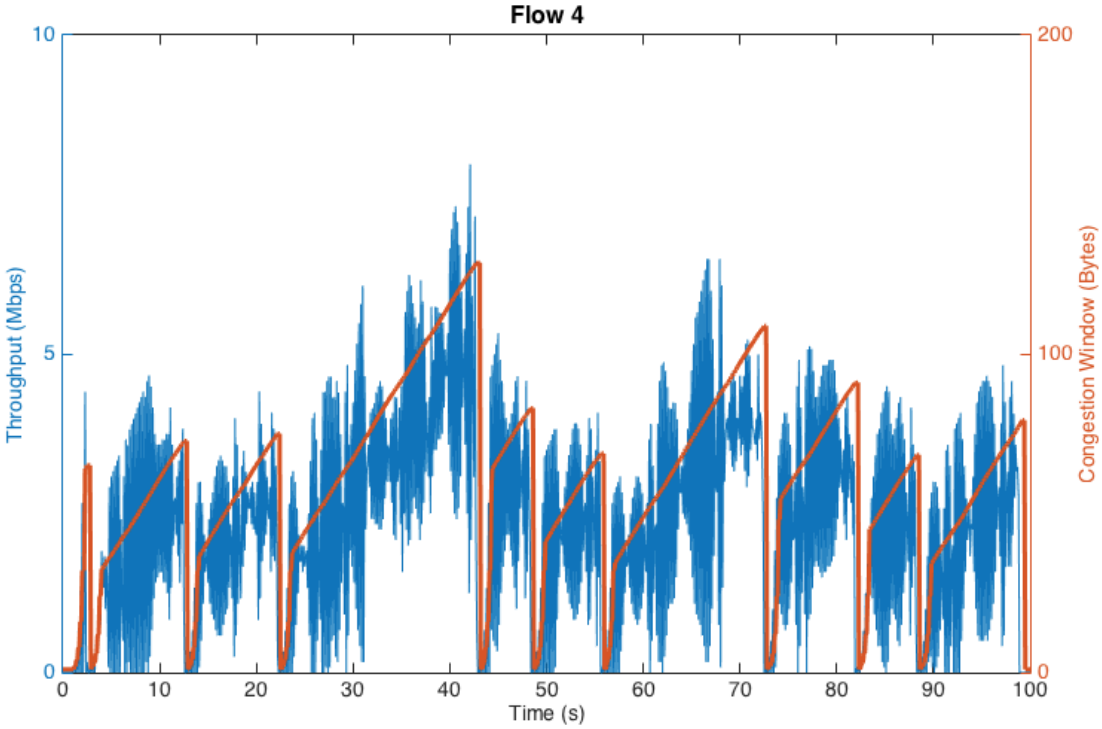

Flow 4
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
5
10
0
100
200
0
10
20
30
40
50
60
70
80
90
100

SACK

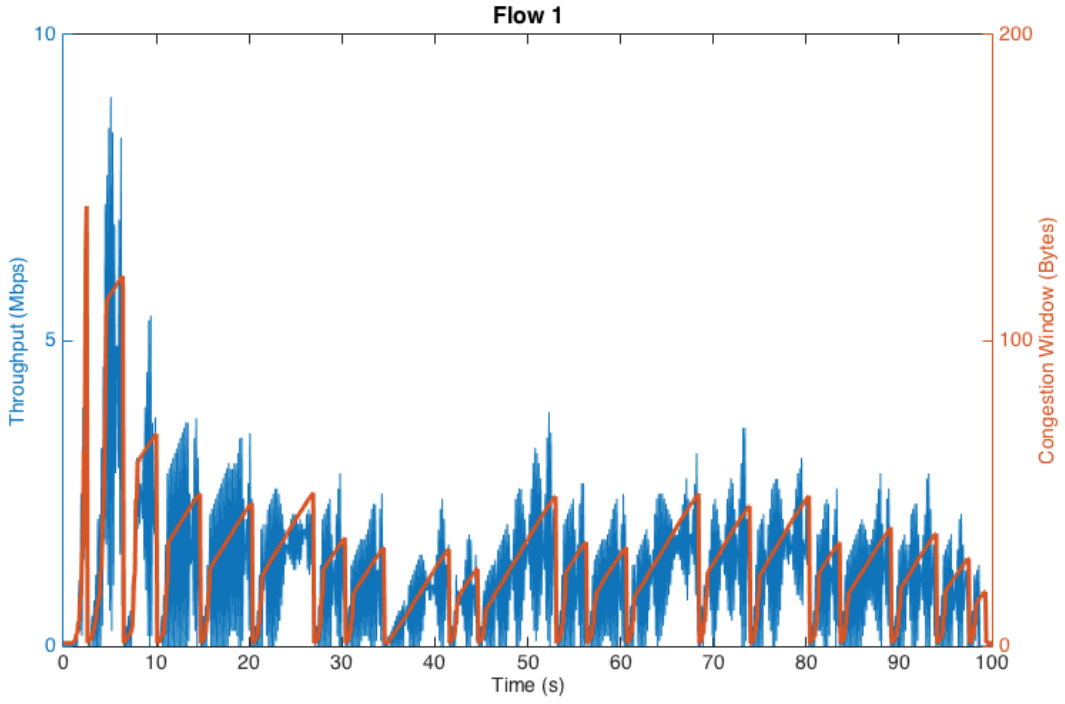

Flow 1
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
2
4
6
8
10
0
50
100
150
200
250
0
10
20
30
40
50
60
70
80
90
100

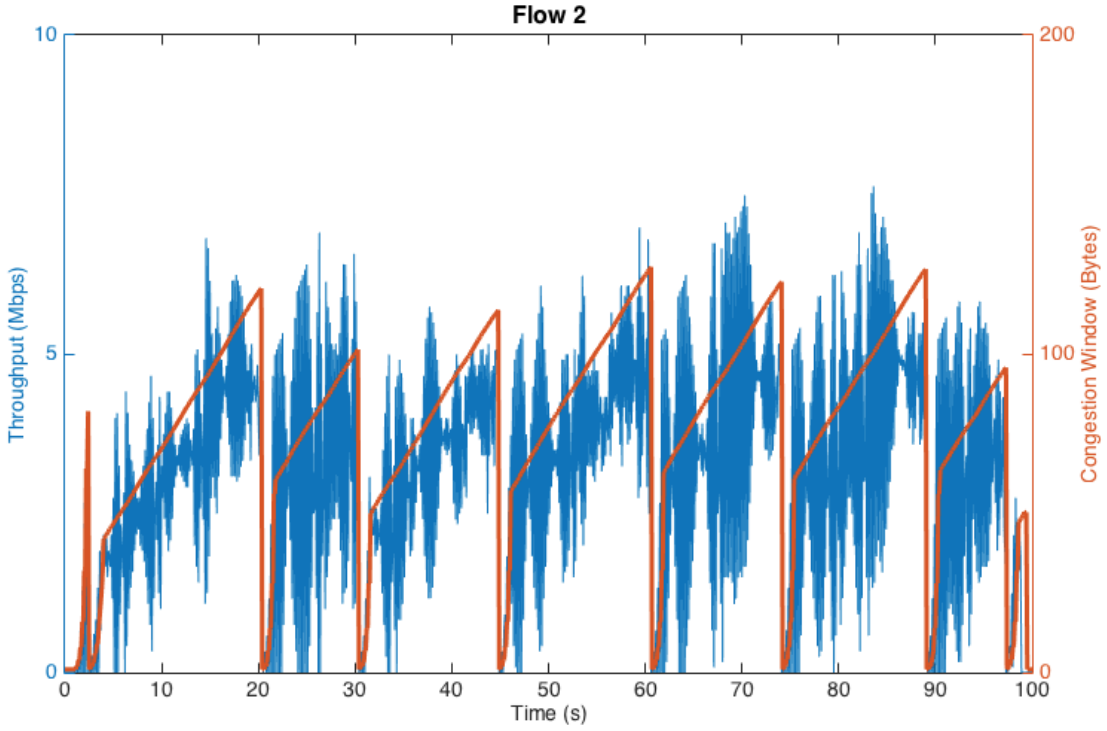

Flow 2
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
5
10
0
100
200
0
10
20
30
40
50
60
70
80
90
100

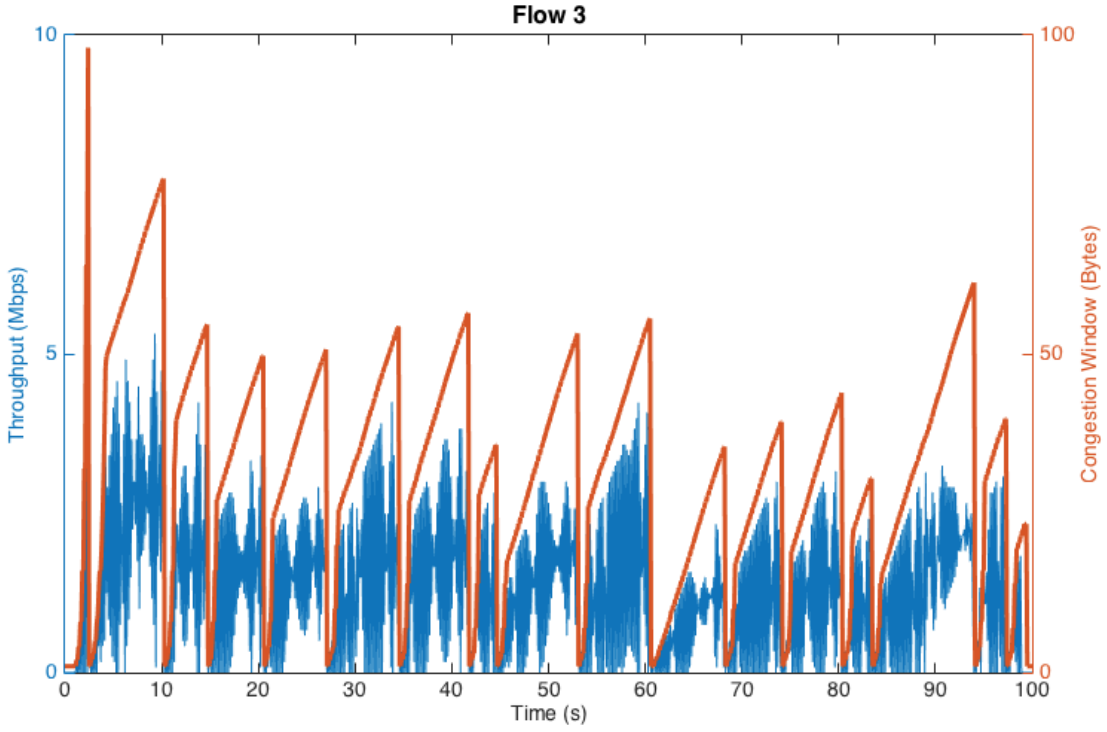

Flow 3
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
1
2
3
4
5
0
20
40
60
80
100
0
10
20
30
40
50
60
70
80
90
100

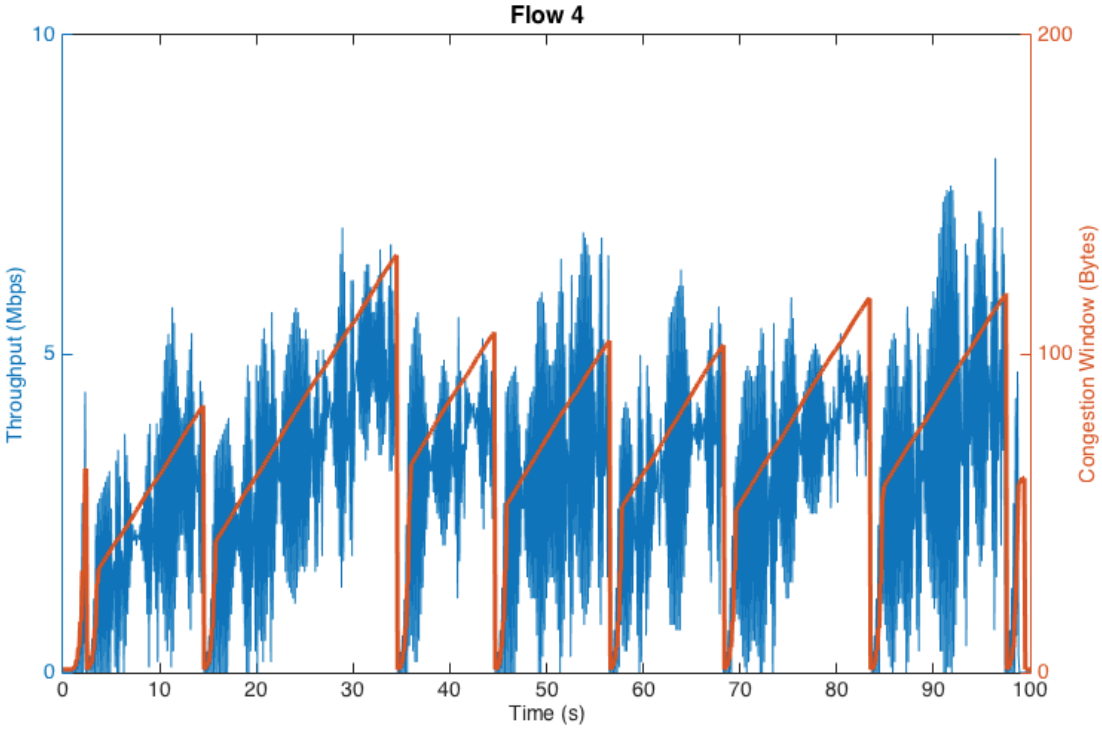

Flow 4
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
5
10
0
100
200
0
10
20
30
40
50
60
70
80
90
100

Tahoe

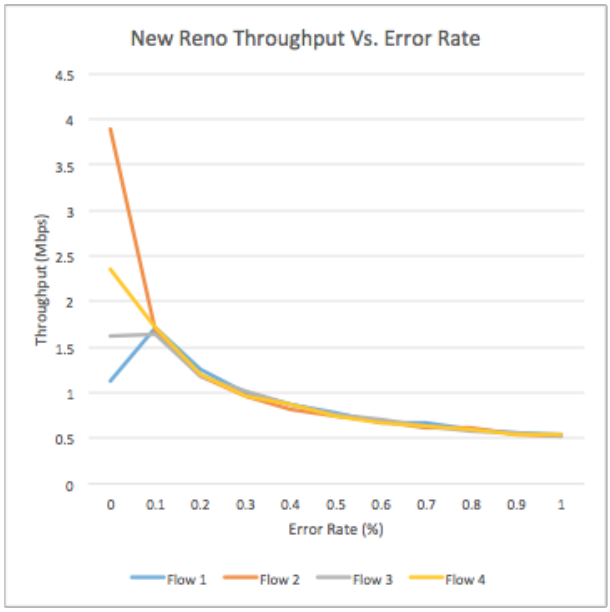

Flow 1
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
5
10
0
100
200
0
10
20
30
40
50
60
70
80
90
100

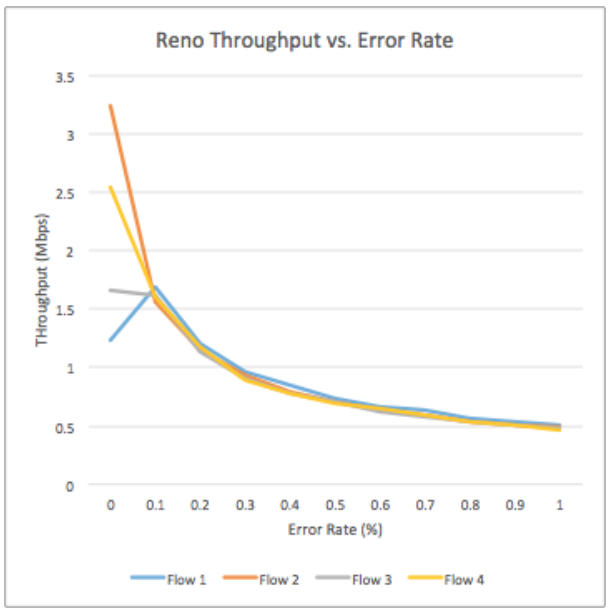

Flow 2
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
5
10
0
100
200
0
10
20
30
40
50
60
70
80
90
100

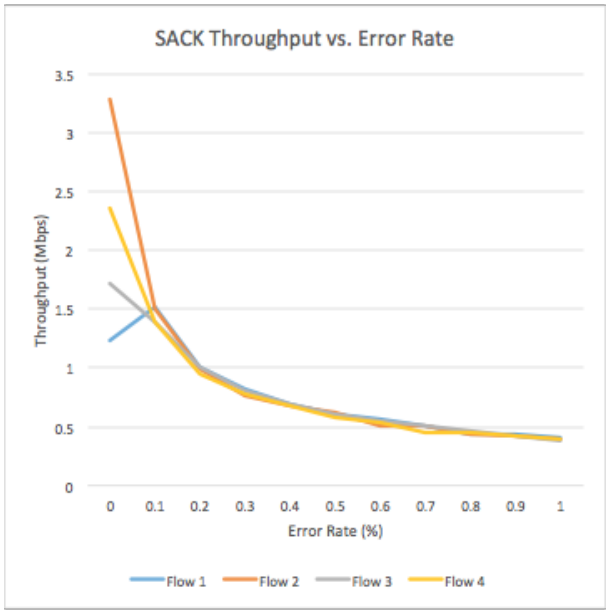

Flow 3
Throughput (Mbps)
Congestion Window (Bytes)
Time (s)
0
5
10
0
50
100
0
10
20
30
40
50
60
70
80
90
100

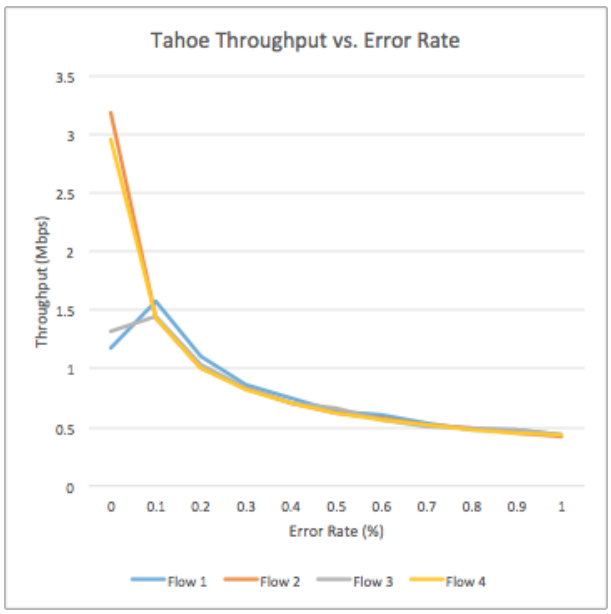


3. A loss module was added into the network between nodes R1 and R2. Average throughput was measured twenty times for each flow for error rates from 0% to 1%. Awk and Shell file are used and shown in Appendix C and D.









4. At zero error rate, Tahoe provides more throughput to Flow 4 between S2-D2 and less throughput to Flow 3 between S2-D1. Meanwhile, new Reno provides less throughput for Flow 4 between S2-D2 and adds more throughput to Flow 2 between S1-D2. As error rate increases beyond 0%, the four TCP flavors begin to perform similarly but with small differences in maximum throughput. At an error rate of 1%, the four New Reno flows each average 0.5289 Mbps. At this same error rate, each Reno flow averages 0.4844 Mbps, each Tahoe flow averages 0.4316 Mbps, and each SACK flow averages 0.3931 Mbps. This means that New Reno performs the best as error rates increase, and SACK performs the worst.

   Tahoe suffers in lossy environments because it requires a full timeout interval to detect a packet loss. Reno adds a Fast Retransmit mode, which detects packet loss earlier by counting duplicate ACKs. New Reno is a slight modification of Reno, which may perform better in lossy environments by refraining from excessive congestion window

reductions. SACK may suffer in high-error environments from overusing channel resources on acknowledgments.

Section 2 - tcpdump output located in Appendix B

1. IP addresses involved:
    a. Local: 192.169.1.187
    b. Remote: 74.125.212.177

2. Port numbers of local and remote hosts:
    a. Local: 4070
    b. Remote: 80

3. Max Segment Size (MSS) used in either direction:
    a. 1260 bytes

4. Total bytes and unique bytes sent by remote end-host:
    a. Total (actual) bytes: 38785361
    b. Unique bytes: 38719841

5. Number of unique data pkts sent by remote end-host:
    a. 30785 pkts

6. Average downlink throughput (by-hand calculation):
    a. 30790 pkts / 111.603 sec = 275.89 pkts / sec

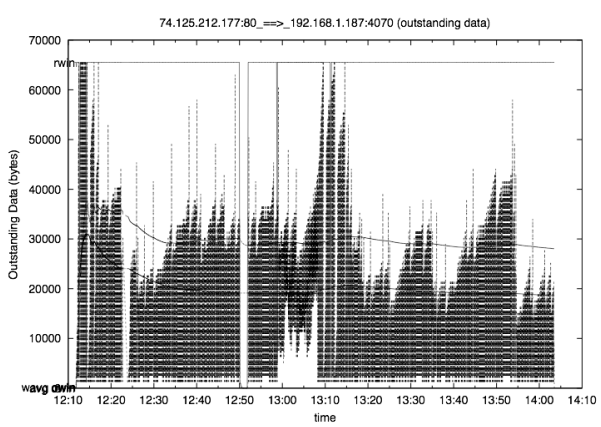
7. Avg RTT in either direction:
    a. Local->Remote: 30.8 ms
    b. Remote->Local: 49.9 ms

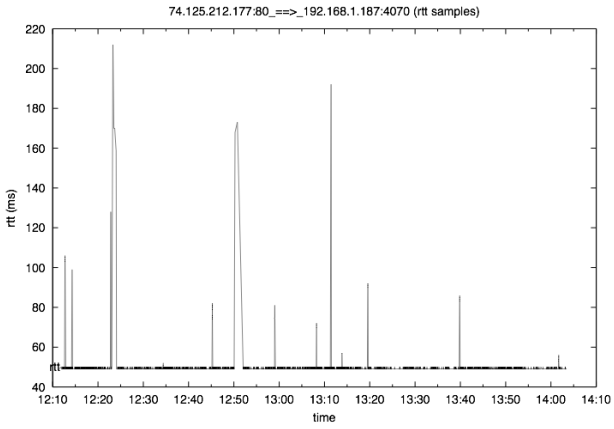
8. Minimum end-to-end RTT:
    a. 30.6 ms

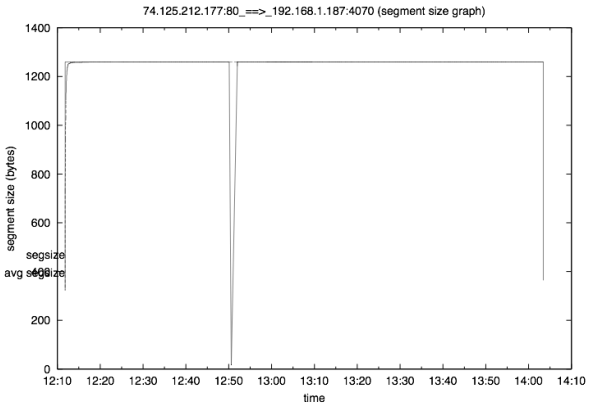
9. TCP flavors used by both end-hosts. How do you identify them?
    a. Local (a): **TCP SACK**. We know this because it uses sacks (see `sack pkts sent` in Appendix B).

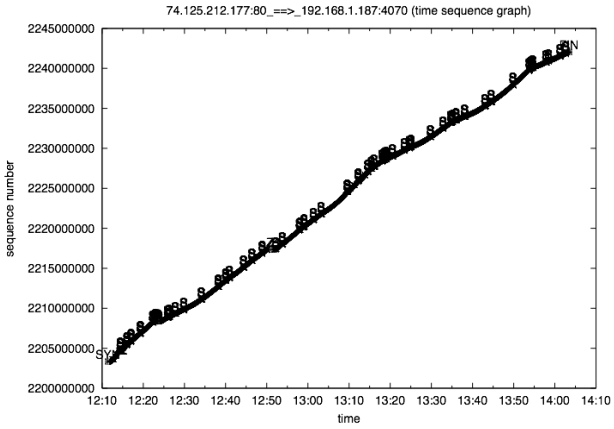
    b. Remote (b): RTT spikes at 12:23, 12:50, 13:12. Look at outstanding window at this point. After 12:20, outstanding data (estimates congestion window at sender) is very low (slow start?). Quickly returns to threshold. Same at 12:50. At 13:12, outstanding data drops and returns nearly immediately to its high window. This could indicate that the fast recovery algorithm is being used. 12:20 and 12:50

definitely indicate slow-start modes, therefore, the remote server uses **TCP Tahoe**.









10. Maximum congestion control window size used by remote end-host:
    a. 8896 bytes

Appendix A

Tcl script for TCP New Reno (other TCP variants, change the `Agent` accordingly):

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

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Blue
$ns color 3 Red
$ns color 4 Red

#Open the NAM trace file
set nf [open projout.nam w]
$ns namtrace-all $nf

#Congestion Window file
set outfile1  [open  "WinFile1"  w]
set outfile2  [open  "WinFile2"  w]
set outfile3  [open  "WinFile3"  w]
set outfile4  [open  "WinFile4"  w]

#Define a 'finish' procedure
proc finish {} {
        global ns nf
        $ns flush-trace
        #Close the NAM trace file
        close $nf
        #Execute NAM on the trace file
        exec nam projout.nam &
        exit 0
}

#Create seven nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 20Mb 4ms DropTail
$ns duplex-link $n1 $n2 10Mb 5ms DropTail
$ns duplex-link $n2 $n3 20Mb 40ms DropTail
$ns duplex-link $n3 $n4 10Mb 45ms DropTail
$ns duplex-link $n4 $n5 20Mb 3ms DropTail
$ns duplex-link $n4 $n6 10Mb 5ms DropTail

#Set Queue Size of all links to 50
$ns queue-limit $n0 $n2 50
$ns queue-limit $n1 $n2 50
$ns queue-limit $n2 $n3 50
$ns queue-limit $n3 $n4 50
$ns queue-limit $n4 $n5 50
$ns queue-limit $n4 $n6 50

#Give node position (for NAM)
```

```
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n3 $n4 orient right
$ns duplex-link-op $n4 $n5 orient right-up
$ns duplex-link-op $n4 $n6 orient right-down

#Setup TCP connection S1-D1
set tcp0 [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n5 $sink0
$ns connect $tcp0 $sink0
$tcp0 set window_ 30000
$tcp0 set fid_ 1

#Setup TCP connection S1-D2
set tcp1 [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n6 $sink1
$ns connect $tcp1 $sink1
$tcp1 set window_ 30000
$tcp1 set fid_ 2

#Setup TCP connection S2-D1
set tcp2 [new Agent/TCP/Newreno]
$ns attach-agent $n1 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n5 $sink2
$ns connect $tcp2 $sink2
$tcp2 set window_ 30000
$tcp2 set fid_ 3

#Setup TCP connection S2-D2
set tcp3 [new Agent/TCP/Newreno]
$ns attach-agent $n1 $tcp3
set sink3 [new Agent/TCPSink]
$ns attach-agent $n6 $sink3
$ns connect $tcp3 $sink3
$tcp3 set window_ 30000
$tcp3 set fid_ 4

#Setup a FTP over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set type_ FTP
$ftp0 set packet_size_ 1500

#Setup a FTP over TCP connection
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
$ftp1 set packet_size_ 1500

#Setup a FTP over TCP connection
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP
$ftp2 set packet_size_ 1500
```

```
#Setup a FTP over TCP connection
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$ftp3 set type_ FTP
$ftp3 set packet_size_ 1500

  proc plotWindow {tcpSource outfile} {
     global ns

     set now [$ns now]
     set cwnd [$tcpSource set cwnd_]

  ###Print TIME CWND   for  gnuplot to plot progressing on CWND
     puts  $outfile  "$now $cwnd"

     $ns at [expr $now+0.1] "plotWindow $tcpSource  $outfile"
  }

set rng [new RNG]
$rng seed 0

# create a random variable that follows the uniform distribution
set loss_random_variable [new RandomVariable/Uniform]
$loss_random_variable use-rng $rng
$loss_random_variable set min_ 0
# the range of the random variable;
$loss_random_variable set max_ 100
set loss_module [new ErrorModel]
# create the error model;
$loss_module drop-target [new Agent/Null]
#a null agent where the dropped packets go to
$loss_module set rate_ 0
# right now rate is 0
# error rate will then be (0.1 = 10 / (100 - 0));
$loss_module ranvar $loss_random_variable
# attach the random variable to loss module;
$ns lossmodel $loss_module $n2 $n3

#Schedule events for the CBR and FTP agents
$ns at 0.0  "plotWindow $tcp0  $outfile1"
$ns at 0.0  "plotWindow $tcp1  $outfile2"
$ns at 0.0  "plotWindow $tcp2  $outfile3"
$ns at 0.0  "plotWindow $tcp3  $outfile4"
$ns at 1.0 "$ftp0 start"
$ns at 1.0 "$ftp1 start"
$ns at 1.0 "$ftp2 start"
$ns at 1.0 "$ftp3 start"
$ns at 99.0 "$ftp0 stop"
$ns at 99.0 "$ftp1 stop"
$ns at 99.0 "$ftp2 stop"
$ns at 99.0 "$ftp3 stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 100.0 "finish"

#Run the simulation
$ns run
```

Appendix B

tcpdump Output:

```
root@kali:~/Desktop/4607 Project# tcptrace -rwl tcp_trace.pcap
1 arg remaining, starting with 'tcp_trace.pcap'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

46880 packets seen, 46880 TCP packets traced
elapsed wallclock time: 0:00:00.137358, 341297 pkts/sec analyzed
trace file elapsed time: 0:01:52.150733
TCP connection info:
1 TCP connection traced:
TCP connection 1:
      host a:        192.168.1.187:4070
      host b:        74.125.212.177:80
      complete conn: yes
      first packet:  Mon Aug 29 16:12:11.356700 2011
      last packet:   Mon Aug 29 16:14:03.507433 2011
      elapsed time:  0:01:52.150733
      total packets: 46880
      filename:      tcp_trace.pcap
   a->b:                        b->a:
     total packets:         16090           total packets:         30790
     ack pkts sent:         16089           ack pkts sent:         30790
     pure acks sent:        16086           pure acks sent:            4
     sack pkts sent:         1466           sack pkts sent:            0
     dsack pkts sent:           0           dsack pkts sent:           0
     max sack blks/ack:         2           max sack blks/ack:         0
     unique bytes sent:      1515           unique bytes sent:  38719841
     actual data pkts:          2           actual data pkts:      30785
     actual data bytes:      1515           actual data bytes:  38785361
     rexmt data pkts:           0           rexmt data pkts:          52
     rexmt data bytes:          0           rexmt data bytes:      65520
     zwnd probe pkts:           0           zwnd probe pkts:           0
     zwnd probe bytes:          0           zwnd probe bytes:          0
     outoforder pkts:           0           outoforder pkts:          16
     pushed data pkts:          1           pushed data pkts:        578
     SYN/FIN pkts sent:       1/1           SYN/FIN pkts sent:       1/1
     req 1323 ws/ts:          Y/N           req 1323 ws/ts:          Y/N
     adv wind scale:            1           adv wind scale:            6
     req sack:                  Y           req sack:                  Y
     sacks sent:             1466           sacks sent:                0
     urgent data pkts:          0 pkts      urgent data pkts:          0 pkts
     urgent data bytes:         0 bytes     urgent data bytes:         0 bytes
     mss requested:          1260 bytes     mss requested:          1380 bytes
     max segm size:          1260 bytes     max segm size:          1260 bytes
     min segm size:           255 bytes     min segm size:            16 bytes
     avg segm size:           757 bytes     avg segm size:          1259 bytes
     max win adv:           65536 bytes     max win adv:            8896 bytes
     min win adv:              16 bytes     min win adv:            8896 bytes
     zero win adv:              3 times     zero win adv:              0 times
     avg win adv:           65251 bytes     avg win adv:            8896 bytes
     initial window:         1515 bytes     initial window:         2197 bytes
     initial window:            2 pkts      initial window:            3 pkts
     ttl stream length:      1515 bytes     ttl stream length:  38719841 bytes
     missed data:               0 bytes     missed data:               0 bytes
     truncated data:         1443 bytes     truncated data:     37677121 bytes
     truncated packets:         2 pkts      truncated packets:     30784 pkts
     data xmit time:        0.000 secs      data xmit time:      111.603 secs
```

idletime max:	708.8 ms	idletime max:	708.9 ms
throughput:	14 Bps	throughput:	345248 Bps
RTT samples:	3	RTT samples:	14596
RTT min:	30.6 ms	RTT min:	49.1 ms
RTT max:	31.1 ms	RTT max:	212.6 ms
RTT avg:	30.8 ms	RTT avg:	49.9 ms
RTT stdev:	0.3 ms	RTT stdev:	3.6 ms
RTT from 3WHS:	31.1 ms	RTT from 3WHS:	49.8 ms
RTT full_sz smpls:	2	RTT full_sz smpls:	1
RTT full_sz min:	30.6 ms	RTT full_sz min:	49.8 ms
RTT full_sz max:	31.1 ms	RTT full_sz max:	49.8 ms
RTT full_sz avg:	30.9 ms	RTT full_sz avg:	49.8 ms
RTT full_sz stdev:	0.0 ms	RTT full_sz stdev:	0.0 ms
post-loss acks:	0	post-loss acks:	52
segs cum acked:	1	segs cum acked:	16086
duplicate acks:	2	duplicate acks:	1435
triple dupacks:	0	triple dupacks:	52
max # retrans:	0	max # retrans:	1
min retr time:	0.0 ms	min retr time:	79.9 ms
max retr time:	0.0 ms	max retr time:	158.0 ms
avg retr time:	0.0 ms	avg retr time:	88.1 ms
sdv retr time:	0.0 ms	sdv retr time:	22.5 ms

Appendix C

```
#============================= throughput.awk ========================

BEGIN {
packetbuffer_flow1 = 0;
packetbuffer_flow2 = 0;
packetbuffer_flow3 = 0;
packetbuffer_flow4 = 0;
}
#body
{
                  event = $1
             time = $3
             Src_node = $5
             Dest_node = $7
            Pkt_type = $9
             Size = $11
            Flow_id = $13



#============= CALCULATE throughput================
#FLOW 1 4-5 S1-D1
if (( event == "r") && ( Pkt_type == "tcp" ) && ( Src_node == "4" ) && ( Dest_node ==
"5" ) && (Flow_id == "1") )
{
flow1_recv++;
packetbuffer_flow1 = packetbuffer_flow1 + Size;
}
#FLOW 2 4-6 S1-D2
if (( event == "r") && ( Pkt_type == "tcp" ) && ( Src_node == "4" ) && ( Dest_node ==
"6" ) && (Flow_id == "2"))
{
flow2_recv++;
packetbuffer_flow2 = packetbuffer_flow2 + Size;
}
#FLOW 3 4-5 S2-D1
if (( event == "r") && ( Pkt_type == "tcp" ) && ( Src_node == "4" ) && ( Dest_node ==
"5" ) && (Flow_id == "3") )
{
flow3_recv++;
packetbuffer_flow3 = packetbuffer_flow3 + Size;
}
#FLOW 4 4-6 S2-D2
if (( event == "r") && ( Pkt_type == "tcp" ) && ( Src_node == "4" ) && ( Dest_node ==
"6" ) && (Flow_id == "4") )
{
flow4_recv++;
packetbuffer_flow4 = packetbuffer_flow4 + Size;
}
} #body


END {
print (packetbuffer_flow1 * 8.0)/((time-1)*10^6), (packetbuffer_flow2 *
8.0)/((time-1)*10^6), (packetbuffer_flow3 * 8.0)/((time-1)*10^6), (packetbuffer_flow4
* 8.0)/((time-1)*10^6);
}
#============================= Ends ============================
```

Appendix D

```
#!/bin/sh
# Replace with
# $rng seed RRPP
# $loss_module set rate_ REPLACEHERE
#
# Script to automate the ECE4607 project
#
echo "Starting Simulations"


echo "NewReno TCP"

for i in  0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
do
       echo "Rate = "$i
       sed 's/REPLACEHERE/'$i'/g' NewReno.tcl > NewReno_mod.tcl
       for j in 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
       do
       sed 's/RRPP/'$j'/g' NewReno_mod.tcl
              ns NewReno_mod.tcl
              awk -f throughput.awk projout_newreno.nam >> out_newreno.txt
       done
       echo "" >> out_newreno.txt
done

echo "Reno TCP"

for i in  0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
do
       echo "Rate = "$i
       sed 's/REPLACEHERE/'$i'/g' Reno.tcl > Reno_mod.tcl
       for j in 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
       do
       sed 's/RRPP/'$j'/g' Reno_mod.tcl
              ns Reno_mod.tcl
              awk -f throughput.awk projout_reno.nam >> out_reno.txt
       done
       echo "" >> out_reno.txt
done

echo "Sack TCP"

for i in  0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
do
       echo "Rate = "$i
       sed 's/REPLACEHERE/'$i'/g' Sack.tcl > Sack_mod.tcl
       for j in 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
       do
       sed 's/RRPP/'$j'/g' Sack_mod.tcl
              ns Sack_mod.tcl
              awk -f throughput.awk projout_sack.nam >> out_sack.txt
       done
       echo "" >> out_sack.txt
done

echo "Tahoe TCP"

for i in  0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
do
```

```
        echo "Rate = "$i
        sed 's/REPLACEHERE/'$i'/g' Tahoe.tcl > Tahoe_mod.tcl
        for j in 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
        do
        sed 's/RRPP/'$j'/g' Tahoe_mod.tcl
               ns Tahoe_mod.tcl
               awk -f throughput.awk projout_tahoe.nam >> out_tahoe.txt
        done
        echo "" >> out_tahoe.txt
done
```