ADVANCE COMPUTER NETWORK LAB.

(CSPC-352)

Submitted to:	Dr. Kunwar Pal
Submitted By:	BALJOT SINGH 18103026
	GAGANDEEP SINGH 18103031
	GAURAV BHARDWAJ 18103034
	JAGNOOR SINGH 18103045
Year:	3rd /6th sem.
Group No:	9
Group:	G2

Performance Comparison of TCP Variants using ns3.

Theory:

TCP stands for **Transmission Control Protocol**. It is a transport layer protocol that facilitates the transmission of packets from source to destination. It is a connection-oriented protocol that means it establishes the connection prior to the communication that occurs between the computing devices in a network.

Features of TCP protocol

- **Transport Layer Protocol**
- **Reliable**
- **Order of the data is maintained**
- **Connection-oriented**
- **Full duplex**
- **Stream-oriented**

TCP Congestion Policy:

- ➢ TCP uses a congestion window and a congestion policy that avoid congestion
- ➢ **Slow Start, Exponential Increase:**
    - This algorithm is based on the idea that the size of the congestion window (cwnd) starts with one maximum segment size (MSS). The MSS is determined during connection establishment by using an option of the same name. The size of the window increases one MSS each time an acknowledgment is received. As the name implies, the window starts slowly, but grows exponentially.
    - Slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named ssthresh (slow-start threshold). When the size of window in bytes reaches this threshold, slow start stops and the next phase starts. In most implementations the value of ssthresh is 65,535 bytes.
- ➢ **Congestion Avoidance, Additive Increase:**
    - If we start with the slow-start algorithm, the size of the congestion window increases exponentially. To avoid congestion before it happens, one must slow down this exponential growth. TCP defines another algorithm called congestion avoidance, which undergoes an additive increase instead of an exponential one.
    - When the size of the congestion window reaches the slow-start threshold, the slow-start phase stops and the additive phase begins. In this algorithm, each time

the whole window of segments is acknowledged (one round), the size of the congestion window is increased by 1.

- **Congestion Detection:**

  - Multiplicative Decrease If congestion occurs, the congestion window size must be decreased. The only way the sender can guess that congestion has occurred is by the need to retransmit a segment.

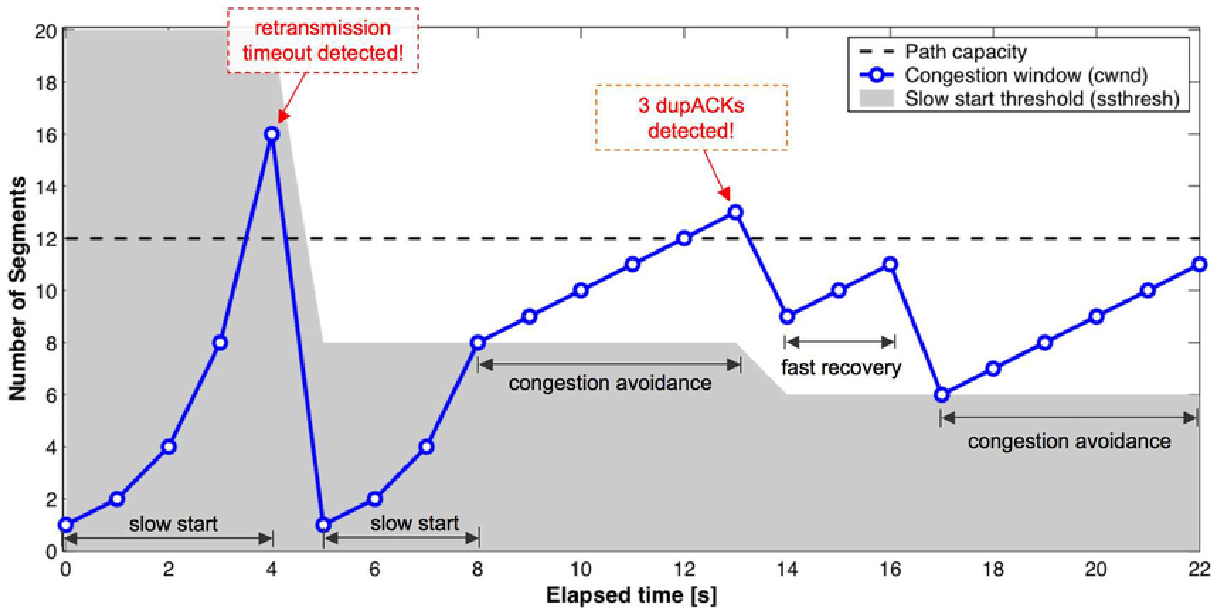
  - Most TCP implementations have two reactions:

➔ If a time-out occurs, there is a stronger possibility of congestion; a segment has probably been dropped in the network, and there is no news about the sent segments.

- In this case TCP reacts strongly:
- It sets the value of the threshold to one-half of the current window size.
- It sets cwnd to the size of one segment.
- It starts the slow-start phase again.

➔ If three ACKs are received, there is a weaker possibility of congestion; a segment may have been dropped, but some segments after that may have arrived safely since three ACKs are received. This is called **fast transmission and fast recovery**.

- In this case, TCP has a weaker reaction:
- It sets the value of the threshold to one-half of the current window size.
- It sets cwnd to the value of the threshold (some implementations add three segment sizes to the threshold).
- It starts the congestion avoidance phase.

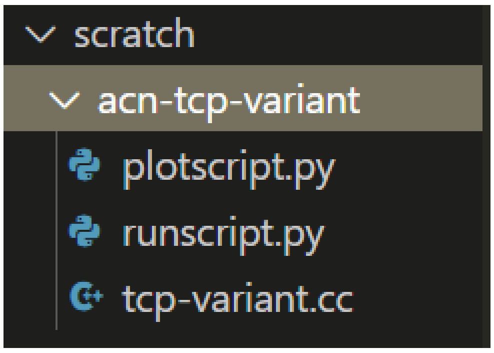


Requirements:

- **Linux system**
- **Ns-3 network simulation.**
- **Python 3**
- **Numpy, Pandas, Matplotlib for graph plotting**

Installation Steps:

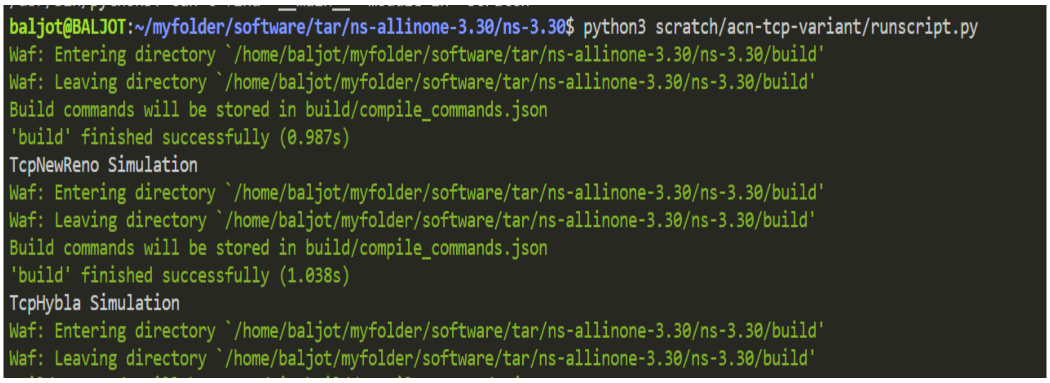
➢ **Go to link and place the folder in scratch directory of ns3**



```
scratch
  acn-tcp-variant
    plotscript.py
    runscript.py
    tcp-variant.cc
```

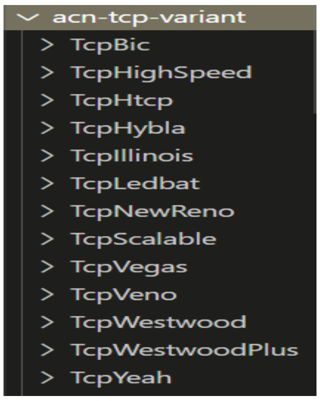
➢ **Navigate to ns3 folder /ns-3.30 and open terminal**

➢ **Run Python3 scratch/acn-tcp-variant/runscript.py**



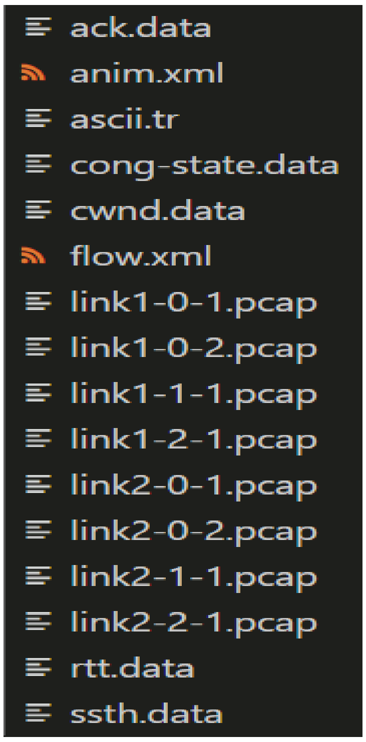
```
baljot@BALJOT:~/myfolder/software/tar/ns-allinone-3.30/ns-3.30$ python3 scratch/acn-tcp-variant/runscript.py
Waf: Entering directory `/home/baljot/myfolder/software/tar/ns-allinone-3.30/ns-3.30/build'
Waf: Leaving directory `/home/baljot/myfolder/software/tar/ns-allinone-3.30/ns-3.30/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.987s)
TcpNewReno Simulation
Waf: Entering directory `/home/baljot/myfolder/software/tar/ns-allinone-3.30/ns-3.30/build'
Waf: Leaving directory `/home/baljot/myfolder/software/tar/ns-allinone-3.30/ns-3.30/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.038s)
TcpHybla Simulation
Waf: Entering directory `/home/baljot/myfolder/software/tar/ns-allinone-3.30/ns-3.30/build'
Waf: Leaving directory `/home/baljot/myfolder/software/tar/ns-allinone-3.30/ns-3.30/build'
```

➢ **After this you can see various folder with name < tcp variant > in acn-tcp-variant folder**



```
acn-tcp-variant
  > TcpBic
  > TcpHighSpeed
  > TcpHtcp
  > TcpHybla
  > TcpIllinois
  > TcpLedbat
  > TcpNewReno
  > TcpScalable
  > TcpVegas
  > TcpVeno
  > TcpWestwood
  > TcpWestwoodPlus
  > TcpYeah
```

- **Content of each <tcp-variant> folder**



```
ack.data
anim.xml
ascii.tr
cong-state.data
cwnd.data
flow.xml
link1-0-1.pcap
link1-0-2.pcap
link1-1-1.pcap
link1-2-1.pcap
link2-0-1.pcap
link2-0-2.pcap
link2-1-1.pcap
link2-2-1.pcap
rtt.data
ssth.data
```

ack.data : acknowledged sequence no with time

ascii.tr: Ascii tracing file

***.pcap:** Packet capture file

anim.xml: Animator file of Visualizing

Flow.xml: Flow Monitor File

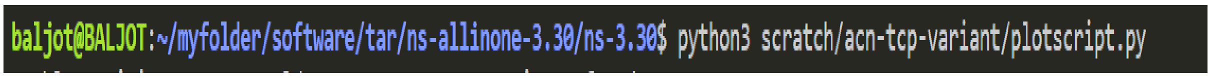
Cwnd.data: Congestion window data file

Ssth.data: slow-start threshold data file

Rtt.data: round-trip time data file

Cong-state.data: Congestion state data file

- **Navigate to ns-3.30 folder and Run** python3 scratch/acn-tcp-variant/plotscript.py

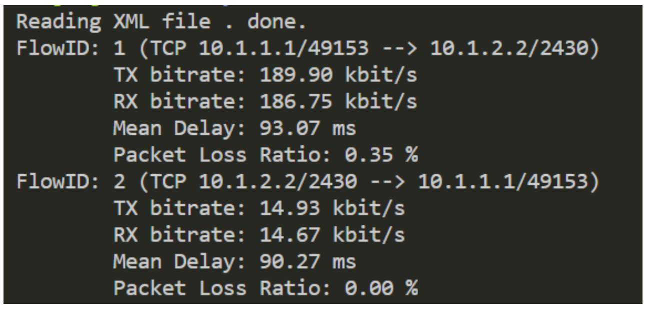


```
baljot@BALJOT:~/myfolder/software/tar/ns-allinone-3.30/ns-3.30$ python3 scratch/acn-tcp-variant/plotscript.py
```

- **Now you can see plot.png in each <tcp-variant> folder and ack.png in ack.png in acn-tcp-variant folder these file will be shown later**

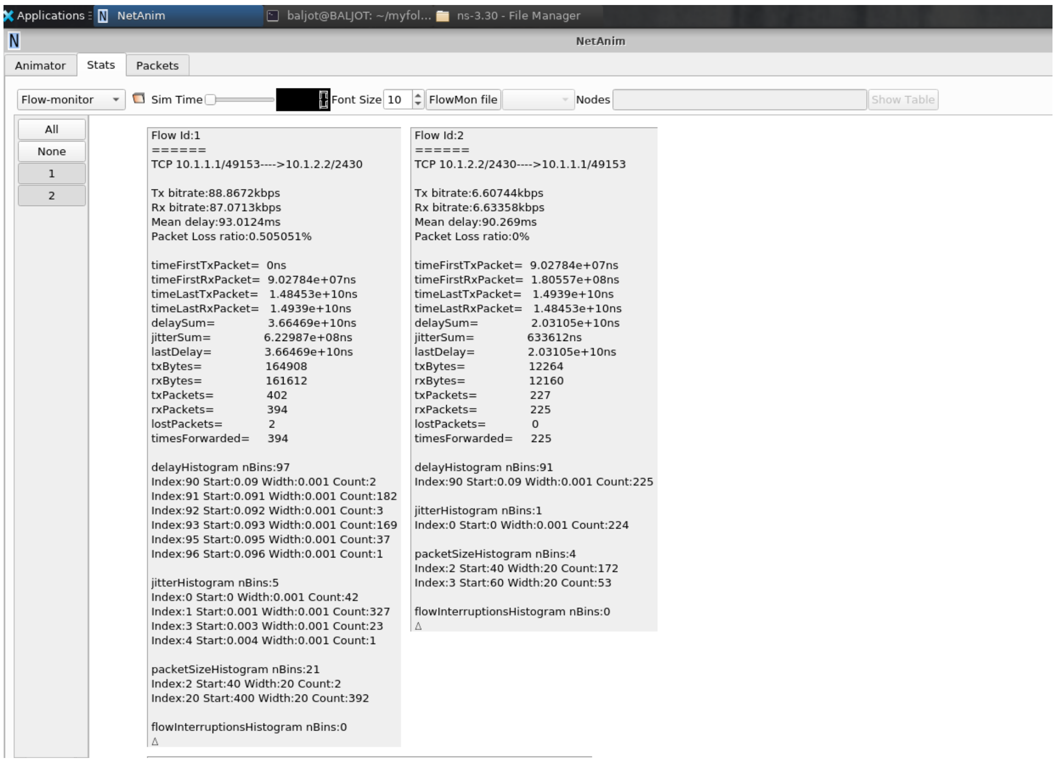
- **To parse flow monitor file either open in netanim and change stats to flow monitor file or copy src/flow-monitor/examples/flowmon-parse-result.py into ns3 folder and Run command** python2 flowmon-parse-result.py scratch/acn-tcp-variant/<name>/flow.xml

Using Terminal



```
Reading XML file . done.
FlowID: 1 (TCP 10.1.1.1/49153 --> 10.1.2.2/2430)
        TX bitrate: 189.90 kbit/s
        RX bitrate: 186.75 kbit/s
        Mean Delay: 93.07 ms
        Packet Loss Ratio: 0.35 %
FlowID: 2 (TCP 10.1.2.2/2430 --> 10.1.1.1/49153)
        TX bitrate: 14.93 kbit/s
        RX bitrate: 14.67 kbit/s
        Mean Delay: 90.27 ms
        Packet Loss Ratio: 0.00 %
```

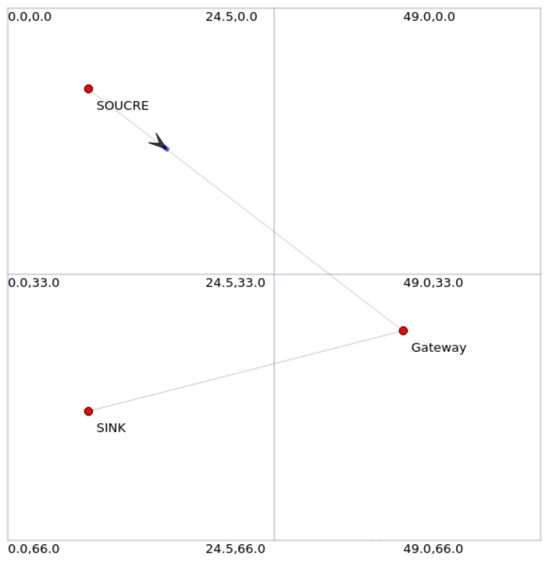
Using NetAnim

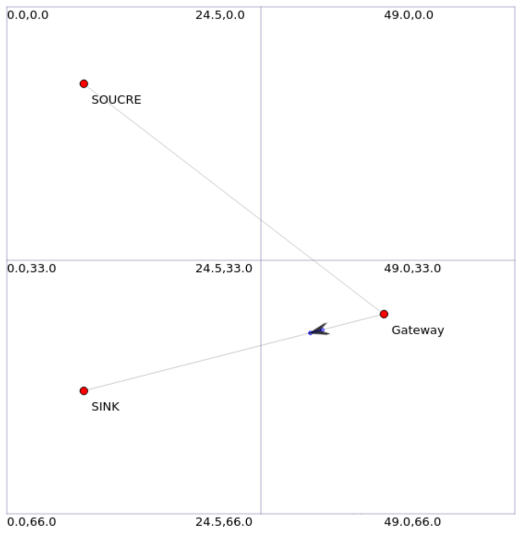


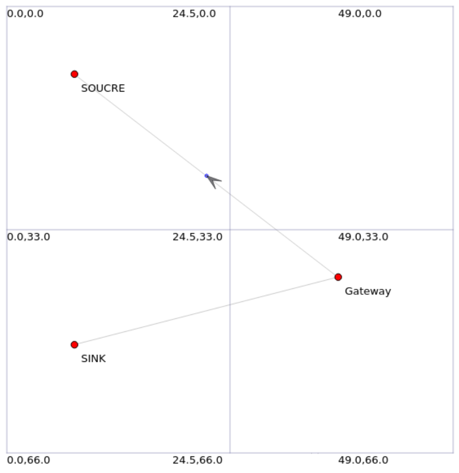
```
Flow Id:1
======
TCP 10.1.1.1/49153---->10.1.2.2/2430

Tx bitrate:88.8672kbps
Rx bitrate:87.0713kbps
Mean delay:93.0124ms
Packet Loss ratio:0.505051%

timeFirstTxPacket=  0ns
timeFirstRxPacket=  9.02784e+07ns
timeLastTxPacket=   1.48453e+10ns
timeLastRxPacket=   1.4939e+10ns
delaySum=           3.66469e+10ns
jitterSum=         6.22987e+08ns
lastDelay=         3.66469e+10ns
txBytes=            164908
rxBytes=            161612
txPackets=          402
rxPackets=          394
lostPackets=         2
timesForwarded=     394

delayHistogram nBins:97
Index:90 Start:0.09 Width:0.001 Count:2
Index:91 Start:0.091 Width:0.001 Count:182
Index:92 Start:0.092 Width:0.001 Count:3
Index:93 Start:0.093 Width:0.001 Count:169
Index:95 Start:0.095 Width:0.001 Count:37
Index:96 Start:0.096 Width:0.001 Count:1

jitterHistogram nBins:5
Index:0 Start:0 Width:0.001 Count:42
Index:1 Start:0.001 Width:0.001 Count:327
Index:3 Start:0.003 Width:0.001 Count:23
Index:4 Start:0.004 Width:0.001 Count:1

packetSizeHistogram nBins:21
Index:2 Start:40 Width:20 Count:2
Index:20 Start:400 Width:20 Count:392

flowInterruptionsHistogram nBins:0
```

```
Flow Id:2
======
TCP 10.1.2.2/2430---->10.1.1.1/49153

Tx bitrate:6.60744kbps
Rx bitrate:6.63358kbps
Mean delay:90.269ms
Packet Loss ratio:0%

timeFirstTxPacket=  9.02784e+07ns
timeFirstRxPacket=  1.80557e+08ns
timeLastTxPacket=   1.4939e+10ns
timeLastRxPacket=   1.48453e+10ns
delaySum=           2.03105e+10ns
jitterSum=         633612ns
lastDelay=         2.03105e+10ns
txBytes=            12264
rxBytes=            12160
txPackets=          227
rxPackets=          225
lostPackets=         0
timesForwarded=     225

delayHistogram nBins:91
Index:90 Start:0.09 Width:0.001 Count:225

jitterHistogram nBins:1
Index:0 Start:0 Width:0.001 Count:224

packetSizeHistogram nBins:4
Index:2 Start:40 Width:20 Count:172
Index:3 Start:60 Width:20 Count:53

flowInterruptionsHistogram nBins:0
```

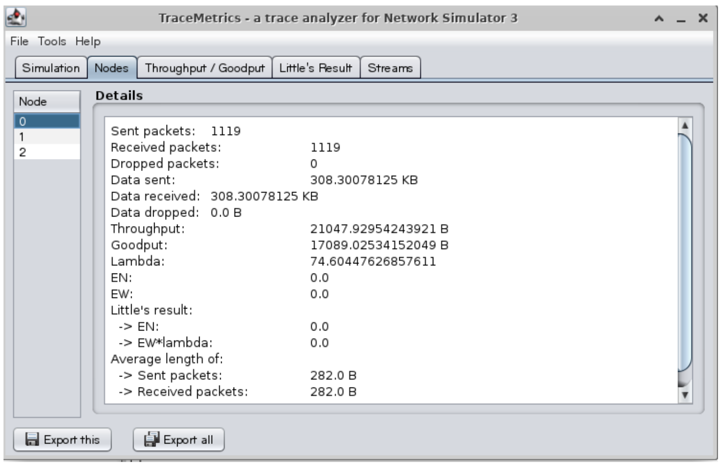
- ➢ **To visualize Use netanim and open anim.xml file in NetAnim**



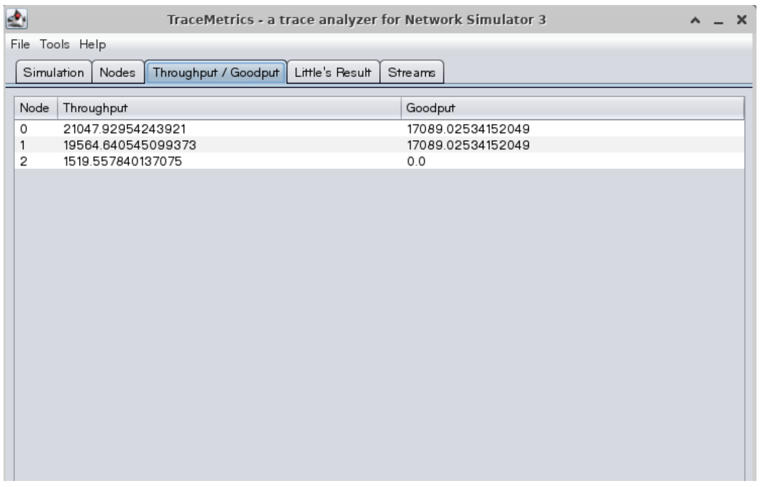




- **To Analyse Ascii trace download tracemetrics and open file in it**



```
Sent packets:    1119
Received packets:               1119
Dropped packets:                0
Data sent:                      308.30078125 KB
Data received:   308.30078125 KB
Data dropped:    0.0 B
Throughput:                     21047.92954243921 B
Goodput:                        17089.02534152049 B
Lambda:                         74.60447626857611
EN:                             0.0
EW:                             0.0
Little's result:
  -> EN:                        0.0
  -> EW*lambda:                 0.0
Average length of:
  -> Sent packets:              282.0 B
  -> Received packets:          282.0 B
```

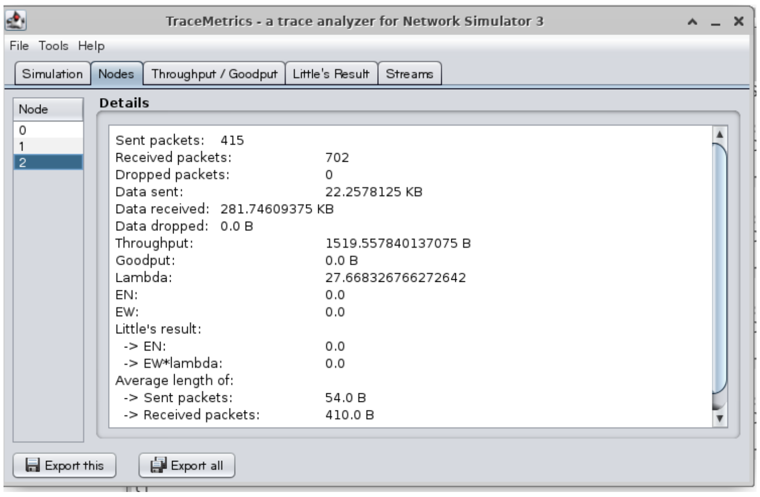


TraceMetrics - a trace analyzer for Network Simulator 3

File Tools Help

Simulation | Nodes | Throughput / Goodput | Little's Result | Streams

Node	Throughput	Goodput
0	21047.92954243921	17089.02534152049
1	19564.640545099373	17089.02534152049
2	1519.557840137075	0.0



TraceMetrics - a trace analyzer for Network Simulator 3

File Tools Help

Simulation | Nodes | Throughput / Goodput | Little's Result | Streams

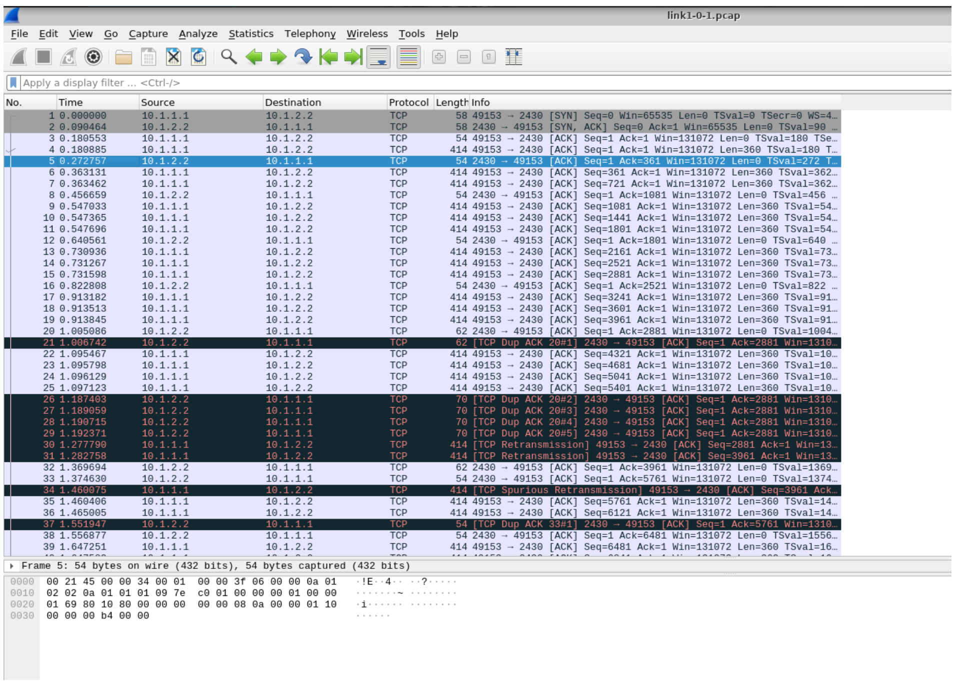
Node: 0, 1, 2

Details

```
Sent packets:    415
Received packets:              702
Dropped packets:               0
Data sent:                     22.2578125 KB
Data received:   281.74609375 KB
Data dropped:    0.0 B
Throughput:                    1519.557840137075 B
Goodput:                       0.0 B
Lambda:                        27.668326766272642
EN:                            0.0
EW:                            0.0
Little's result:
  -> EN:                       0.0
  -> EW*lambda:                0.0
Average length of:
  -> Sent packets:             54.0 B
  -> Received packets:         410.0 B
```

Export this | Export all

➢ **To analyse Pcap file open file in wireshark**



link1-0-1.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

```
No.   Time      Source     Destination  Protocol Length Info
    1 0.000000  10.1.1.1   10.1.2.2     TCP       58 49153 → 2430 [SYN] Seq=0 Win=65535 Len=0 TSval=0 TSecr=0 WS=4…
    2 0.090464  10.1.2.2   10.1.1.1     TCP       58 2430 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 TSval=90 …
    3 0.180553  10.1.1.1   10.1.2.2     TCP       54 49153 → 2430 [ACK] Seq=1 Ack=1 Win=131072 Len=0 TSval=180 TSe…
    4 0.180885  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=1 Ack=1 Win=131072 Len=360 TSval=180 T…
    5 0.272757  10.1.2.2   10.1.1.1     TCP       54 2430 → 49153 [ACK] Seq=1 Ack=361 Win=131072 Len=0 TSval=272 T…
    6 0.363131  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=361 Ack=1 Win=131072 Len=360 TSval=362…
    7 0.363462  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=721 Ack=1 Win=131072 Len=360 TSval=362…
    8 0.456659  10.1.2.2   10.1.1.1     TCP       54 2430 → 49153 [ACK] Seq=1 Ack=1081 Win=131072 Len=0 TSval=456 …
    9 0.547033  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=1081 Ack=1 Win=131072 Len=360 TSval=54…
   10 0.547365  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=1441 Ack=1 Win=131072 Len=360 TSval=54…
   11 0.547696  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=1801 Ack=1 Win=131072 Len=360 TSval=54…
   12 0.640561  10.1.2.2   10.1.1.1     TCP       54 2430 → 49153 [ACK] Seq=1 Ack=1801 Win=131072 Len=0 TSval=640 …
   13 0.730936  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=2161 Ack=1 Win=131072 Len=360 TSval=73…
   14 0.731267  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=2521 Ack=1 Win=131072 Len=360 TSval=73…
   15 0.731598  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=2881 Ack=1 Win=131072 Len=360 TSval=73…
   16 0.822808  10.1.2.2   10.1.1.1     TCP       54 2430 → 49153 [ACK] Seq=1 Ack=2521 Win=131072 Len=0 TSval=822 …
   17 0.913182  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=3241 Ack=1 Win=131072 Len=360 TSval=91…
   18 0.913513  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=3601 Ack=1 Win=131072 Len=360 TSval=91…
   19 0.913845  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=3961 Ack=1 Win=131072 Len=360 TSval=91…
   20 1.005086  10.1.2.2   10.1.1.1     TCP       62 2430 → 49153 [ACK] Seq=1 Ack=2881 Win=131072 Len=0 TSval=1004…
   21 1.006742  10.1.2.2   10.1.1.1     TCP       62 [TCP Dup ACK 20#1] 2430 → 49153 [ACK] Seq=1 Ack=2881 Win=1310…
   22 1.095467  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=4321 Ack=1 Win=131072 Len=360 TSval=10…
   23 1.095798  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=4681 Ack=1 Win=131072 Len=360 TSval=10…
   24 1.096129  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=5041 Ack=1 Win=131072 Len=360 TSval=10…
   25 1.097123  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=5401 Ack=1 Win=131072 Len=360 TSval=10…
   26 1.187403  10.1.2.2   10.1.1.1     TCP       70 [TCP Dup ACK 20#2] 2430 → 49153 [ACK] Seq=1 Ack=2881 Win=1310…
   27 1.189059  10.1.2.2   10.1.1.1     TCP       70 [TCP Dup ACK 20#3] 2430 → 49153 [ACK] Seq=1 Ack=2881 Win=1310…
   28 1.190715  10.1.2.2   10.1.1.1     TCP       70 [TCP Dup ACK 20#4] 2430 → 49153 [ACK] Seq=1 Ack=2881 Win=1310…
   29 1.192371  10.1.2.2   10.1.1.1     TCP       70 [TCP Dup ACK 20#5] 2430 → 49153 [ACK] Seq=1 Ack=2881 Win=1310…
   30 1.277790  10.1.1.1   10.1.2.2     TCP      414 [TCP Retransmission] 49153 → 2430 [ACK] Seq=2881 Ack=1 Win=13…
   31 1.282758  10.1.1.1   10.1.2.2     TCP      414 [TCP Retransmission] 49153 → 2430 [ACK] Seq=3961 Ack=1 Win=13…
   32 1.369694  10.1.2.2   10.1.1.1     TCP       62 2430 → 49153 [ACK] Seq=1 Ack=3961 Win=131072 Len=0 TSval=1369…
   33 1.374630  10.1.2.2   10.1.1.1     TCP       54 2430 → 49153 [ACK] Seq=1 Ack=5761 Win=131072 Len=0 TSval=1374…
   34 1.460075  10.1.1.1   10.1.2.2     TCP      414 [TCP Spurious Retransmission] 49153 → 2430 [ACK] Seq=3961 Ack…
   35 1.460406  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=5761 Ack=1 Win=131072 Len=360 TSval=14…
   36 1.465005  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=6121 Ack=1 Win=131072 Len=360 TSval=14…
   37 1.551947  10.1.2.2   10.1.1.1     TCP       54 [TCP Dup ACK 33#1] 2430 → 49153 [ACK] Seq=1 Ack=5761 Win=1310…
   38 1.556877  10.1.2.2   10.1.1.1     TCP       54 2430 → 49153 [ACK] Seq=1 Ack=6481 Win=131072 Len=0 TSval=1556…
   39 1.647251  10.1.1.1   10.1.2.2     TCP      414 49153 → 2430 [ACK] Seq=6481 Ack=1 Win=131072 Len=360 TSval=16…

▸ Frame 5: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)

0000  00 21 45 00 00 34 00 01  00 00 3f 06 00 00 0a 01   ·!E··4·· ··?·····
0010  02 02 0a 01 01 01 09 7e  c0 01 00 00 00 01 00 00   ·······~ ········
0020  01 69 80 10 80 00 00 00  00 00 08 0a 00 00 01 10   ·i······ ········
0030  00 00 00 b4 00 00                                  ······
```

Code:

Tcp-variant.cc

```
/*

Topology used

┌──────────────┐            ┌──────────────┐             ┌──────────────┐
│ NODE 1       │  link 1    │ NODE 0       │  link 2     │ NODE 2       │
│ SOURCE       ├────────────┤ GATEWAY      ├─────────────┤  SINK        │   │
└──────────────┘            └──────────────┘             └──────────────┘


*/


#include <iostream>
#include <fstream>
#include <string>
#include <sys/stat.h>
#include <sys/types.h>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/error-model.h"
#include "ns3/tcp-header.h"
#include "ns3/udp-header.h"
#include "ns3/enum.h"
#include "ns3/event-id.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/netanim-module.h"
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("TcpVariantsComparison");

bool firstCwnd = true;
bool firstSshThr = true;
bool firstRtt=true;
// output streams
Ptr<OutputStreamWrapper> cWndStream;
Ptr<OutputStreamWrapper> ssThreshStream;
Ptr<OutputStreamWrapper> ackStream;
Ptr<OutputStreamWrapper> congStateStream;
Ptr<OutputStreamWrapper> rttStream;

uint32_t cWndValue = 0;
uint32_t ssThreshValue = 0;
uint32_t rtovalue=0;


static void
CwndTracer (uint32_t oldval, uint32_t newval)
{

  if (firstCwnd)
```

```
    {
      *cWndStream->GetStream () << "0.0 " << oldval << std::endl;
      firstCwnd = false;
    }
  *cWndStream->GetStream () << Simulator::Now ().GetSeconds () << " " << newval << std::endl;
  cWndValue = newval;

  if (!firstSshThr)
    {
      *ssThreshStream->GetStream () << Simulator::Now ().GetSeconds () << " " << ssThreshValue << std::endl;
    }
}

static void
SsThreshTracer (uint32_t oldval, uint32_t newval)
{
  if (firstSshThr)
    {
      *ssThreshStream->GetStream () << "0.0 " << oldval << std::endl;
      firstSshThr = false;
    }
  *ssThreshStream->GetStream () << Simulator::Now ().GetSeconds () << " " << newval << std::endl;
  ssThreshValue = newval;

  if (!firstCwnd)
    {
      *cWndStream->GetStream () << Simulator::Now ().GetSeconds () << " " << cWndValue << std::endl;
    }
}

static void
RttTracer (Time oldval, Time newval)
{
  if (firstRtt)
    {
      *rttStream->GetStream () << "0.0 " << oldval.GetSeconds () << std::endl;
      firstRtt = false;
    }
  *rttStream->GetStream () << Simulator::Now ().GetSeconds () << " " << newval.GetSeconds () << std::endl;
}

static void
CongStateTracer (TcpSocketState::TcpCongState_t old, TcpSocketState::TcpCongState_t newState)
{
  *congStateStream->GetStream () << Simulator::Now ().GetSeconds () << " " << newState << std::endl;
}

static void
AckTracer (SequenceNumber32 old, SequenceNumber32 newAck)
{
  *ackStream->GetStream () << Simulator::Now ().GetSeconds () << " " << newAck << std::endl;
}

static void
TraceCwnd (std::string cwnd_tr_file_name)
{
  AsciiTraceHelper ascii;
  cWndStream  = ascii.CreateFileStream (cwnd_tr_file_name.c_str ());
  Config::ConnectWithoutContext ("/NodeList/1/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow", MakeCallback (&
CwndTracer));
}

static void
TraceSsThresh (std::string ssthresh_tr_file_name)
{
  AsciiTraceHelper ascii;
  ssThreshStream = ascii.CreateFileStream (ssthresh_tr_file_name.c_str ());
  Config::ConnectWithoutContext("/NodeList/1/$ns3::TcpL4Protocol/SocketList/0/SlowStartThreshold", MakeCallback (&
SsThreshTracer));
}
static void
TraceRtt (std::string rtt_tr_file_name)
{
  AsciiTraceHelper ascii;
  rttStream = ascii.CreateFileStream (rtt_tr_file_name.c_str ());
  Config::ConnectWithoutContext ("/NodeList/1/$ns3::TcpL4Protocol/SocketList/0/RTT", MakeCallback (&RttTracer));
}



static void
TraceAck (std::string &ack_file_name)
{
  AsciiTraceHelper ascii;
  ackStream = ascii.CreateFileStream (ack_file_name.c_str ());
  Config::ConnectWithoutContext ("/NodeList/1/$ns3::TcpL4Protocol/SocketList/0/HighestRxAck", MakeCallback (&AckT
racer));
}

static void
TraceCongState (std::string &cong_state_file_name)
{
  AsciiTraceHelper ascii;
  congStateStream = ascii.CreateFileStream (cong_state_file_name.c_str ());
  Config::ConnectWithoutContext ("/NodeList/1/$ns3::TcpL4Protocol/SocketList/0/CongState", MakeCallback (&CongStat
eTracer));
}

int main (int argc, char *argv[])
{
  std::string transport_prot = "TcpWestwood";
  double error_p = 0.0;
  std::string bandwidth = "2Mbps";
  std::string delay = "0.01ms";
  std::string access_bandwidth = "10Mbps";
  std::string access_delay = "45ms";

  double data_mbytes = 0;
  uint32_t mtu_bytes = 400;
  float duration = 15;
  uint32_t run = 0;
  uint32_t cbr=0;


  CommandLine cmd;
  cmd.AddValue ("transport_prot", "Transport protocol to use: TcpNewReno, TcpLinuxReno, "
                  "TcpHybla, TcpHighSpeed, TcpHtcp, TcpVegas, TcpScalable, TcpVeno, "
                  "TcpBic, TcpYeah, TcpIllinois, TcpWestwood, TcpWestwoodPlus, TcpLedbat", transport_prot);
  cmd.AddValue ("error_p", "Packet error rate", error_p);
  cmd.AddValue ("bandwidth", "Bottleneck bandwidth", bandwidth);
  cmd.AddValue ("delay", "Bottleneck bandwidth", delay);
  cmd.AddValue ("access_bandwidth", "Access link bandwidth", access_bandwidth);
  cmd.AddValue ("access_delay", "Access link delay", access_delay);
  cmd.AddValue ("data", "Number of Megabytes of data to transmit", data_mbytes);
  cmd.AddValue ("mtu", "Size of IP packets to send in bytes", mtu_bytes);
  cmd.AddValue("cbr","No of cbr traffic genrator",cbr);

  cmd.Parse (argc, argv);
  NS_LOG_UNCOND(transport_prot+" Simulation");
  std::string file="scratch/acn-tcp-variant/"+transport_prot;
  mkdir(file.c_str(), 0777);
  SeedManager::SetSeed (1);
  SeedManager::SetRun (run);

  // User may find it convenient to enable logging
  LogComponentEnable("TcpVariantsComparison", LOG_LEVEL_ALL);


  // Calculate the ADU size
  Header* temp_header = new Ipv4Header ();
  uint32_t ip_header = temp_header->GetSerializedSize ();
  //NS_LOG_LOGIC ("IP Header size is: " << ip_header);
  delete temp_header;
  temp_header = new TcpHeader ();
  uint32_t tcp_header = temp_header->GetSerializedSize ();
  //NS_LOG_LOGIC ("TCP Header size is: " << tcp_header);
  delete temp_header;
  uint32_t tcp_adu_size = mtu_bytes - (ip_header + tcp_header);
  //NS_LOG_LOGIC ("TCP ADU size is: " << tcp_adu_size);

  Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (tcp_adu_size));


  // Set the simulation start and stop time
  float start_time = 0.0;
  float stop_time = start_time + duration;
  transport_prot = std::string ("ns3::") + transport_prot;

  // Select TCP variant
   if (transport_prot.compare ("ns3::TcpWestwoodPlus") == 0)
     {
       // TcpWestwoodPlus is not an actual TypeId name; we need TcpWestwood here
       Config::SetDefault ("ns3::TcpL4Protocol::SocketType", TypeIdValue (TcpWestwood::GetTypeId ()));
       // the default protocol type in ns3::TcpWestwood is WESTWOOD
       Config::SetDefault ("ns3::TcpWestwood::ProtocolType", EnumValue (TcpWestwood::WESTWOODPLUS));
     }
   else
     {
       TypeId tcpTid;
       Config::SetDefault ("ns3::TcpL4Protocol::SocketType", TypeIdValue (TypeId::LookupByName (transport_prot)));
     }

  // Create gateways, sources, and sinks
  NodeContainer gateways;
```

```
gateways.Create (1);
NodeContainer sources;
sources.Create (1);
NodeContainer sinks;
sinks.Create (1);


// Configure the error model
// Here we use RateErrorModel with packet error rate
Ptr<UniformRandomVariable> uv = CreateObject<UniformRandomVariable> ();
uv->SetStream (50);
RateErrorModel error_model;
error_model.SetRandomVariable (uv);
error_model.SetUnit (RateErrorModel::ERROR_UNIT_PACKET);
error_model.SetRate (error_p);




InternetStackHelper stack;
stack.InstallAll ();



// Configure the sources and sinks net devices
// and the channels between the sources/sinks and the gateways
PointToPointHelper link1;
link1.SetDeviceAttribute ("DataRate", StringValue (access_bandwidth));
link1.SetChannelAttribute ("Delay", StringValue (access_delay));

PointToPointHelper link2;
link2.SetDeviceAttribute ("DataRate", StringValue (bandwidth));
link2.SetChannelAttribute ("Delay", StringValue (access_delay));
link2.SetDeviceAttribute ("ReceiveErrorModel", PointerValue (&error_model));




NetDeviceContainer devices1,devices2;
devices1 = link1.Install (sources.Get (0), gateways.Get (0));
devices2=link2.Install(gateways.Get (0), sinks.Get (0));

Ipv4AddressHelper address;



Ipv4InterfaceContainer interfaces1,interfaces2;

address.SetBase ("10.1.1.0", "255.255.255.0");
interfaces1 = address.Assign (devices1);
address.SetBase ("10.1.2.0", "255.255.255.0");
interfaces2= address.Assign (devices2);



//NS_LOG_INFO ("Initialize Global Routing.");
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

```
uint16_t port = 2430;

Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory", sinkLocalAddress);
sinkHelper.SetAttribute ("Protocol", TypeIdValue (TcpSocketFactory::GetTypeId ()));


AddressValue remoteAddress (InetSocketAddress (interfaces2.GetAddress (1), port));

// source traffic genrator
BulkSendHelper source ("ns3::TcpSocketFactory", Address ());



source.SetAttribute ("Remote", remoteAddress);
source.SetAttribute ("SendSize", UintegerValue (tcp_adu_size));
// 0 means unlimited
source.SetAttribute ("MaxBytes", UintegerValue (int(data_mbytes * 1000000)));


ApplicationContainer sourceApp = source.Install (sources.Get (0));

sourceApp.Start (Seconds (start_time));
sourceApp.Stop (Seconds (stop_time));



ApplicationContainer sinkApp = sinkHelper.Install (sinks);
sinkApp.Start (Seconds (start_time));
sinkApp.Stop (Seconds (stop_time));

// traffic genrator installed in node 0
ApplicationContainer cbrApps;
ApplicationContainer cbrSinkApps;
uint32_t cbrPort=6759;

for(uint32_t i=0; i<cbr; i++)
  {

    OnOffHelper onOffHelper ("ns3::UdpSocketFactory", InetSocketAddress (interfaces2.GetAddress (1), cbrPort));
    onOffHelper.SetAttribute ("PacketSize", UintegerValue (1024));
    onOffHelper.SetAttribute ("DataRate", StringValue ("1Mbps"));
    cbrApps.Add (onOffHelper.Install (gateways.Get (0)));
    cbrApps.Start (Seconds (7));
    cbrApps.Stop (Seconds (10));


  }
  PacketSinkHelper sink ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), cbrPort));
  cbrSinkApps.Add(sink.Install (sinks.Get (0)));
  cbrSinkApps.Start (Seconds (5));
  cbrSinkApps.Stop (Seconds (10));


// Set up tracing
AsciiTraceHelper ascii;
stack.EnableAsciiIpv4All (ascii.CreateFileStream(file+"/ascii.tr"));
Simulator::Schedule (Seconds (0.00001), &TraceCwnd, file + "/cwnd.data");
```

```
  Simulator::Schedule (Seconds (0.00001), &TraceSsThresh, file + "/ssth.data");
  Simulator::Schedule (Seconds (0.00001), &TraceAck, file + "/ack.data");
  Simulator::Schedule (Seconds (0.00001), &TraceCongState, file + "/cong-state.data");
  Simulator::Schedule (Seconds (0.00001), &TraceRtt, file + "/rtt.data");

  link2.EnablePcapAll (file+"/link2", true);
  link1.EnablePcapAll (file+"/link1", true);

  // Flow monitor
  FlowMonitorHelper flowHelper;
  flowHelper.InstallAll ();

  //net anim
  AnimationInterface anim(file+"/anim.xml"); //to save file for netAnim
 anim.UpdateNodeDescription(sources.Get(0),"SOUCRE");

 anim.UpdateNodeDescription(gateways.Get(0),"Gateway");

 anim.UpdateNodeDescription(sinks.Get(0),"SINK");

  Simulator::Stop (Seconds (stop_time));
  Simulator::Run ();
  flowHelper.SerializeToXmlFile (file+"/flow.xml", true, true);
  Simulator::Destroy ();
  return 0;
}
```

PLOTSCRIPT.PY

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
tcp_variants=[ "TcpNewReno","TcpWestwood","TcpVeno" ,"TcpHybla","TcpHighSpeed","TcpHtcp","TcpVegas", "TcpSca
lable","TcpBic", "TcpYeah", "TcpLedbat","TcpIllinois", "TcpWestwoodPlus" ]

segment = 360




def get_data(type,protocol):
    data=pd.read_csv(f"scratch/acn-tcp-variant/{protocol}/{type}.data",sep="\s+",names=[0,1],engine='python')

    return data.T.to_numpy()


def wrappper(cwnd,statedata):
   xaxis=[]
   yaxis=[]
   counter=0
   state=0;
   for x in cwnd[0]:
       if counter!=len(statedata[0]) and statedata[0][counter]>=x:
          state=statedata[1][counter]
          counter+=1
```

```
        xaxis.append(x)
        yaxis.append(state)

    return statedata


def plot(protocol):
    plt.figure(1,figsize=(12,12))
    plt.tight_layout()

    plt.subplot(2,1,1)
    cwnddata=get_data("cwnd",protocol)
    ssthdata=get_data("ssth",protocol)
    statedata=get_data("cong-state",protocol)
    plt.plot(cwnddata[0], cwnddata[1] / segment,color='darkblue', label='cwnd',linewidth=2)
    plt.plot(ssthdata[0], ssthdata[1] / segment,color='red', linestyle='dotted', label='ssth',linewidth=2)

    plt.ylim(0, cwnddata[1] .max()/segment + 20)
    plt.ylabel('segment(1seg=360bytes)')
    plt.xlabel('Time(sec)')

    fill_data=wrappper(cwnddata,statedata)
    plt.axvspan(0,15,facecolor='lightblue', alpha=0.1)
    for i in range(0,len(fill_data[0])-1):
        if fill_data[1][i] == 1: #disorder
            plt.axvspan(fill_data[0][i],fill_data[0][i+1]+0.1,facecolor='green', alpha=0.5)
        elif fill_data[1][i]== 3: #recovery
           plt.axvspan(fill_data[0][i],fill_data[0][i+1]+0.1,facecolor='yellow', alpha=0.5)
        elif fill_data[1][i]== 4: #Loss
            plt.axvspan(fill_data[0][i],fill_data[0][i+1]+0.1,facecolor='red', alpha=0.5)

    plt.plot([],[],color='lightblue',label='open')
    plt.plot([],[],color='yellow',label='recovery')
    plt.plot([],[],color='green',label='disorder')
    plt.plot([],[],color='red',label='loss')
    plt.title("CONGESTION WINDOW VS SLOW START THRESHOLD")
    plt.legend()

    plt.subplot(2,1,2)
    plt.subplots_adjust(wspace=2)
    rttdata=get_data("rtt",protocol)
    plt.plot(rttdata[0],rttdata[1])
    plt.title("ROUND TRIP TIME")
    plt.ylabel('RTT')
    plt.xlabel('Time(sec)')


    plt.suptitle(protocol)


    plt.savefig(f'scratch/acn-tcp-variant/{protocol}/plot.png')
    plt.close()



def plotack(protocol):
    plt.figure(500)
    data=get_data("ack",protocol)
    plt.plot(data[0],data[1] / segment,label=protocol)




if __name__ == '__main__':

    for protocol in tcp_variants:
        plot(protocol)
        plotack(protocol)
    plt.figure(500,figsize=(10,10))
    plt.title("ACKNOWLEDGEMENT")
    plt.xlabel("TIME(ns)")
    plt.ylabel("SEGMENT(360bytes)")
    plt.legend()
    plt.savefig(f'scratch/acn-tcp-variant/ack-plot.png')
    plt.close()
    print("Potting done")
```

RUNSCRIPT.PY

```
    #running tcp variants one by one
import os

tcp_variants=["TcpNewReno", "TcpHybla", "TcpHighSpeed","TcpHtcp","TcpVegas", "TcpScalable","TcpVeno" ,"TcpBic",
 "TcpYeah", "TcpIllinois", "TcpWestwood", "TcpWestwoodPlus", "TcpLedbat"]
for protocol in tcp_variants:
    os.system(f'./waf --run "scratch/acn-tcp-variant/acn-tcp-variant --transport_prot={protocol} --error_p=0.00 --cbr=0" ')
```

NS3-CONGESTION-STATE

- **OPEN** : This is the so-called normal state. Some algorithms have two phases: Slow start (SS) and Congestion avoidance (CA).if cwnd is smaller than ssthresh, it corresponds to the **Slow start phase**, and if it is larger, it corresponds to the **Congestion avoidance phase**.

- **DISORDER** : A state in which a duplicate ACK (Duplicate ACK) has been received. There is a possibility of packet loss or disorder of reception order.

- **RECOVERY** : A state in which a triple duplicate ACK has been received. We are confident of packet loss and start resending.

- **LOSS** : RTT becomes larger than the retransmission time out (RTO), that is, the ACK timeout is detected. There may be serious congestion.

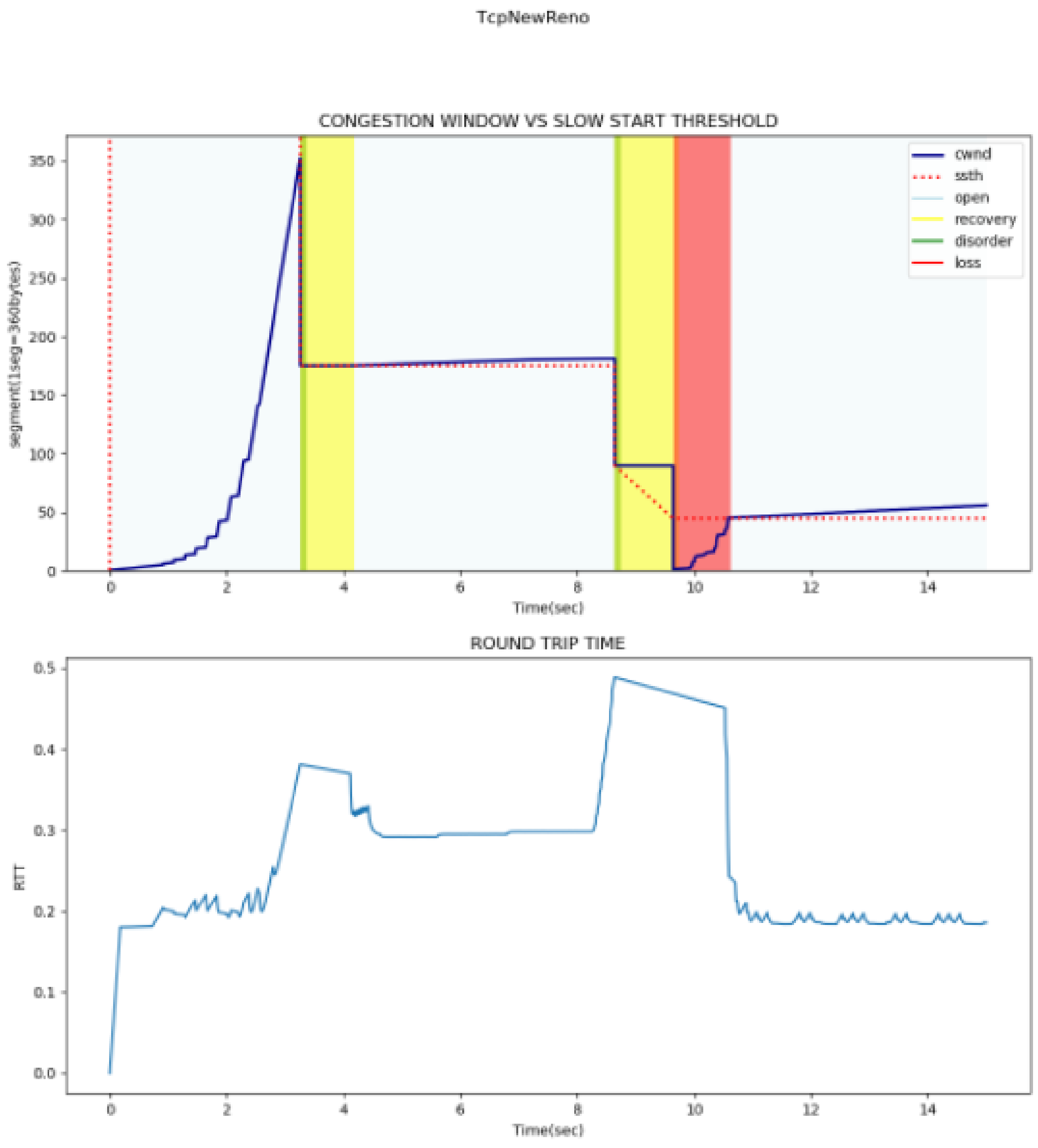
TCP VARIANTS USED IN NS3 SIMULATION

Note: To see every congestion state we have used 5 constant bit rate traffic generator between 7 to 10 seconds to cause serious congestion state in link2

- **NewReno**

TCP New Reno is efficient as compare to Tahoe and Reno. In Reno Fast Recovery exits when three duplicate acknowledgements are received but in New Reno it does not exist until all outstanding data is acknowledged or ends when retransmission timeout occurs. In this way it avoids unnecessary multiple fast retransmit from single window data.

Plot:

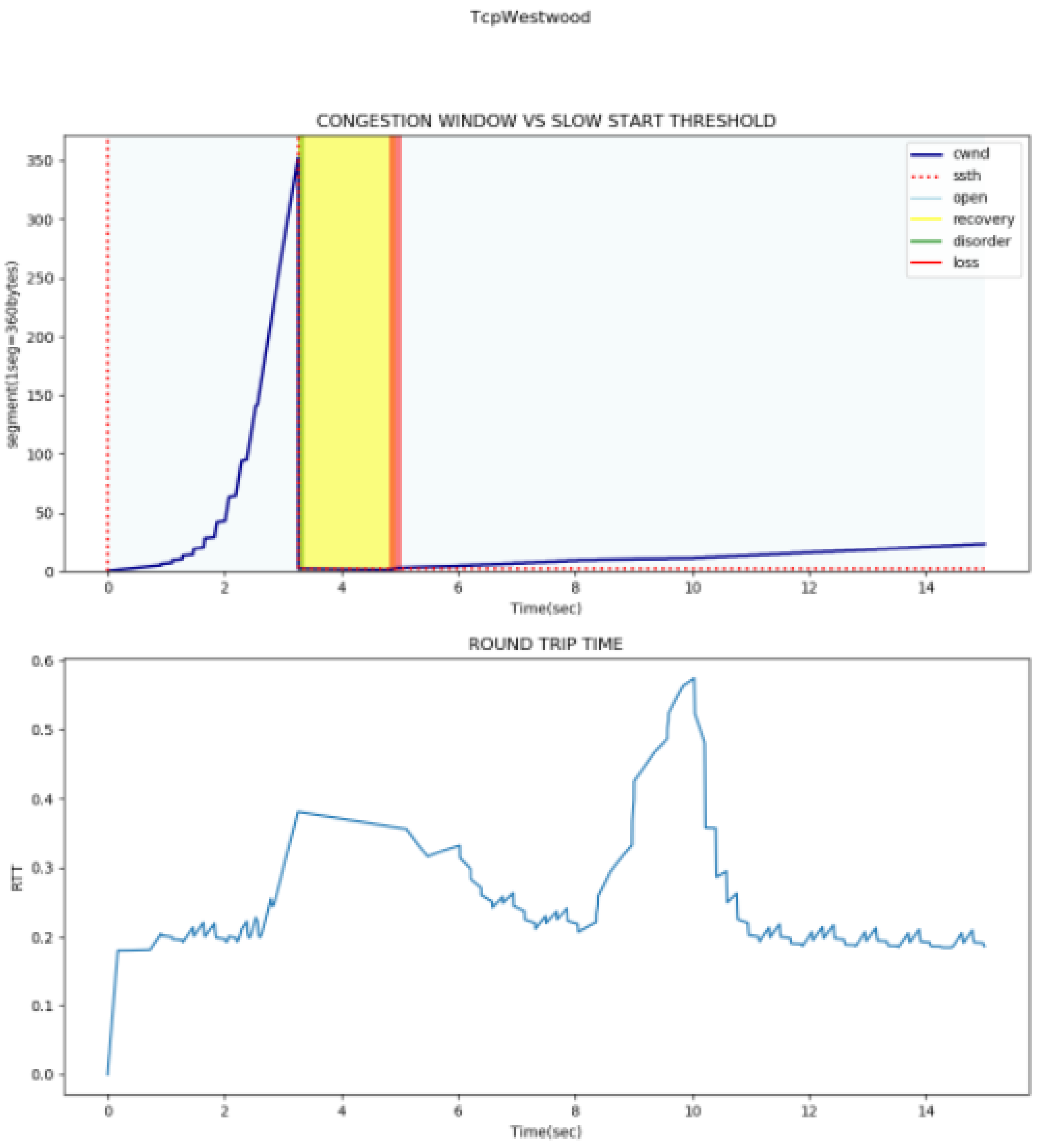


➢ Westwood

TCP Westwood (TCPW) is a sender-side-only modification to TCP New Reno that is intended to better handle large bandwidth-delay product paths (large pipes), with potential packet loss due to transmission or other errors (leaky pipes), and with dynamic load (dynamic pipes).[1]

TCP Westwood relies on mining the ACK stream for information to help it better set the congestion control parameters: Slow Start Threshold (ssthresh), and Congestion Window (cwin).

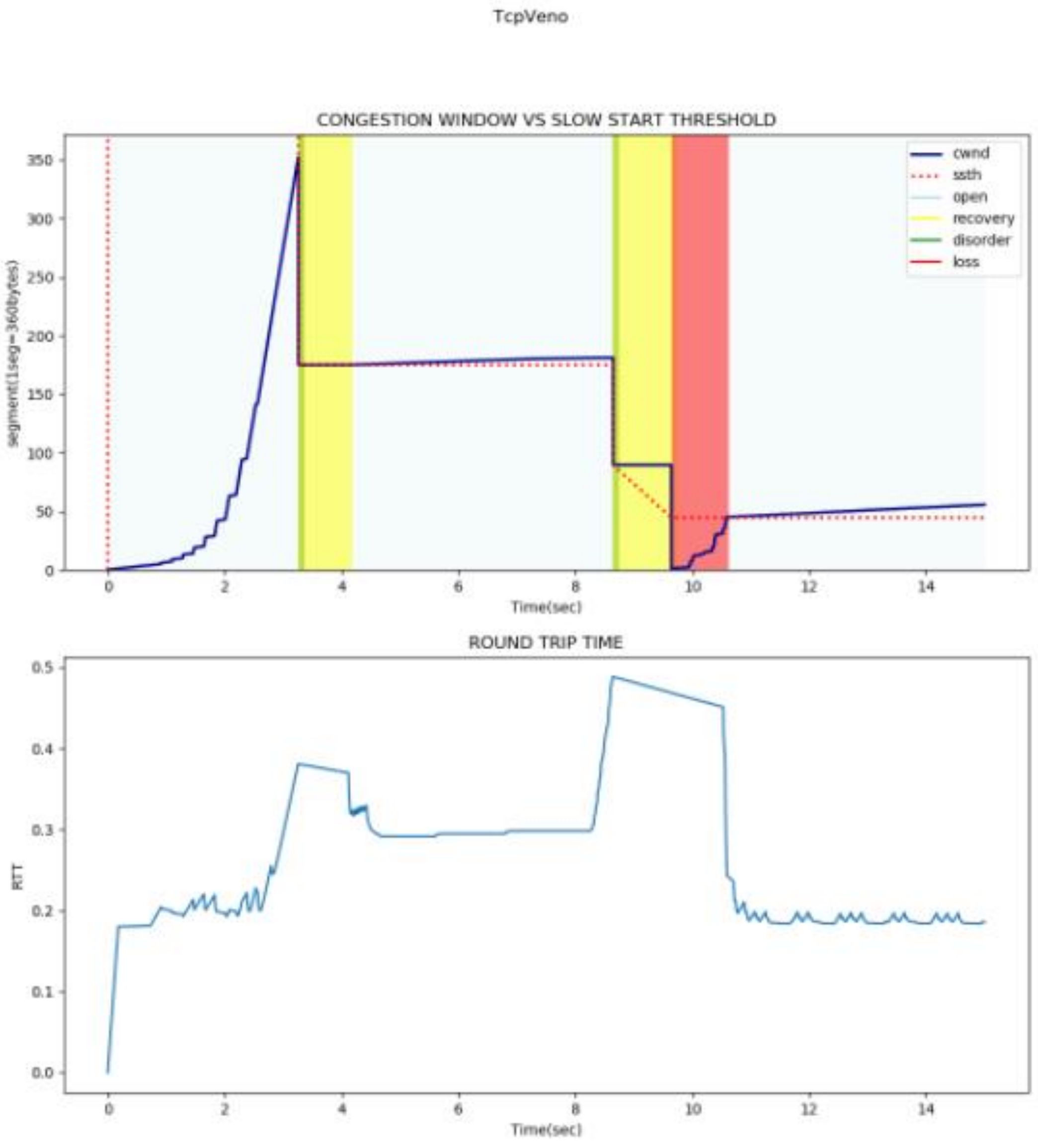
Plot:



➢ **Veno**

TCP Veno that is simple and effective for dealing with random packet loss. A key ingredient of Veno is that it monitors the network congestion level and uses that information to decide whether packet losses are likely to be due to congestion or random bit errors**.**

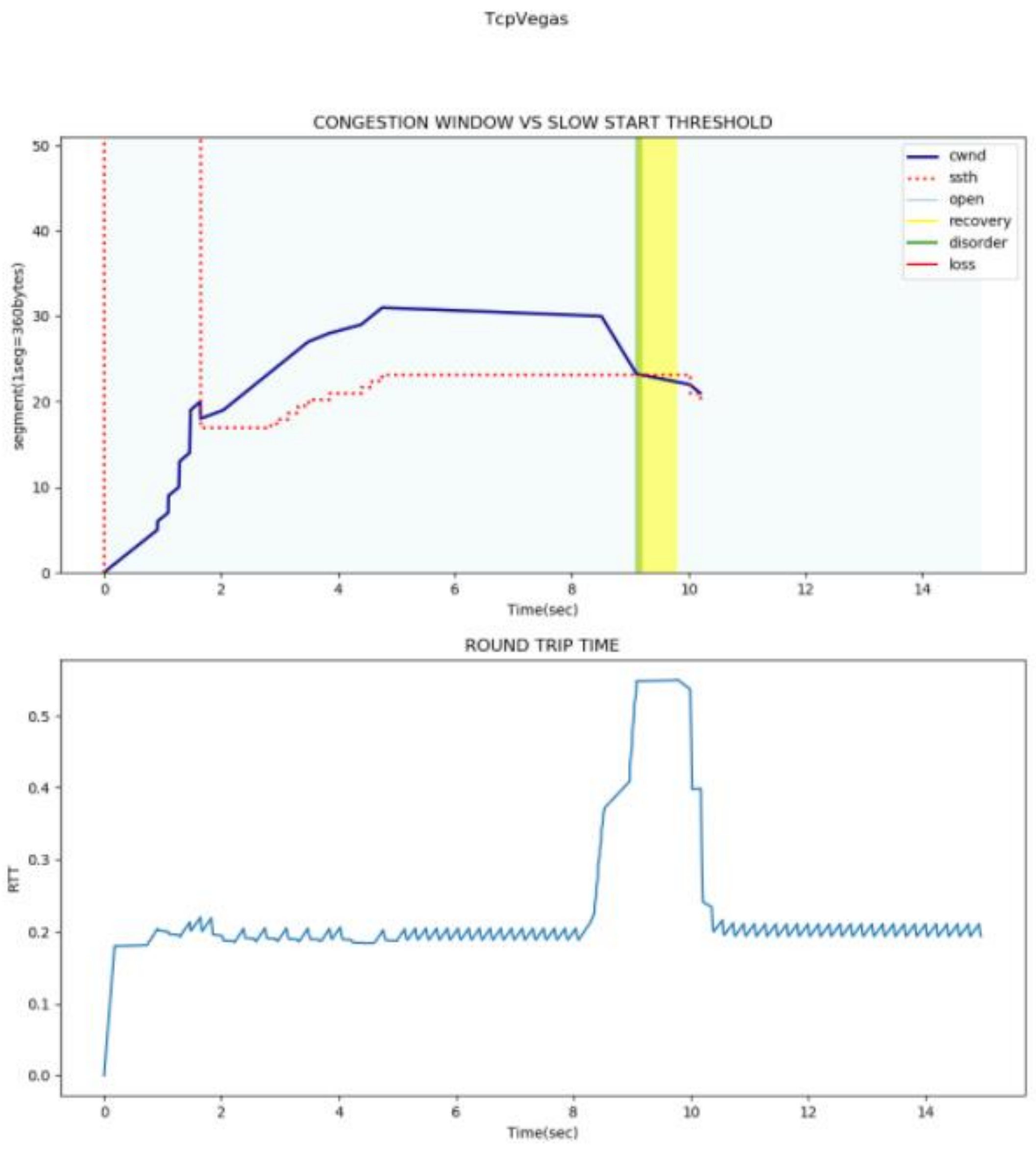
Plot:



➢ Vegas

TCP Vegas detects congestion at an incipient stage based on increasing Round-Trip Time (RTT) values of the packets in the connection unlike other flavors such as Reno, New Reno, etc., which detect congestion only after it has actually happened via packet loss. The algorithm depends heavily on accurate calculation of the Base RTT value. If it is too small then throughput of the connection will be less than the bandwidth available while if the value is too large then it will overrun the connection.

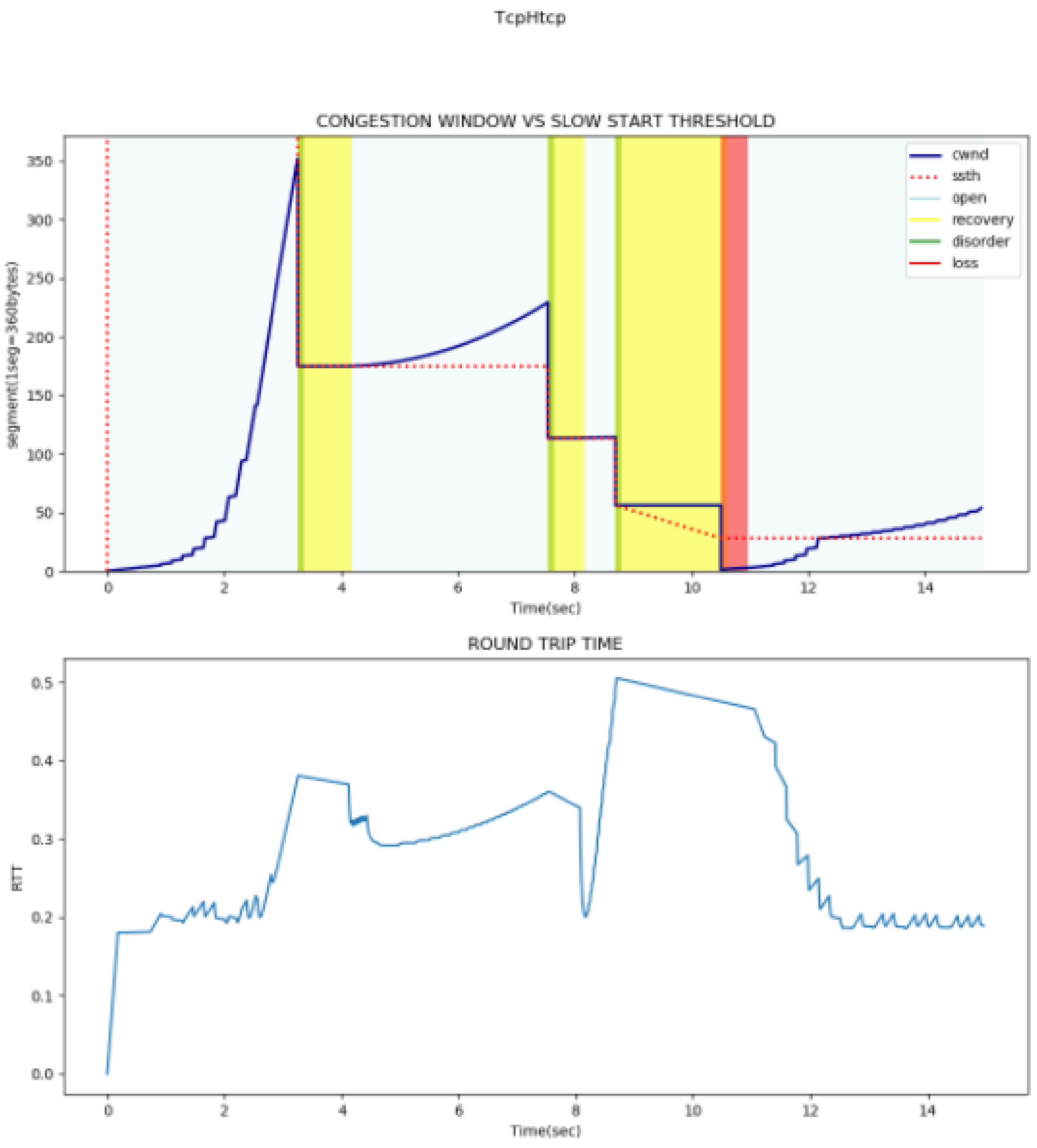
Plot:



➢ HTCP

H-TCP is another implementation of TCP with an optimized congestion control algorithm for high speed networks with high latency.H-TCP is a loss-based algorithm, using additive-increase/multiplicative-decrease (AIMD) to control TCP's congestion window.H-TCP increases its aggressiveness (in particular, the rate of additive increase) as the time since the previous loss increases.

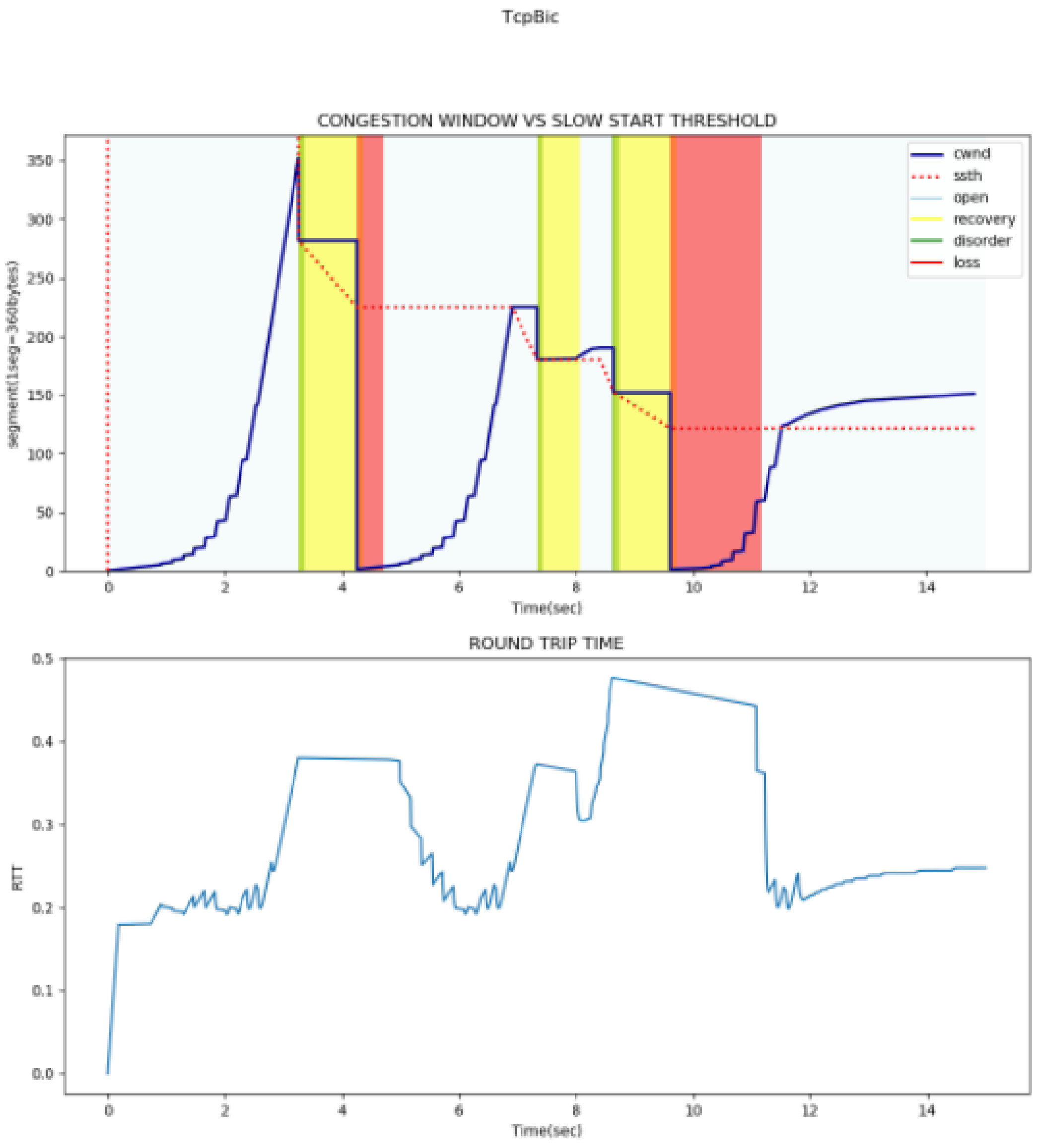
Plot:



- **BIC**

Binary Increase Congestion control (BIC) is a TCP implementation with an optimized CCA for high speed networks with high latency,
BIC implements a unique congestion window (cwnd) algorithm. This algorithm tries to find the maximum cwnd by searching in three parts: binary search increase, additive increase, and slow start. When a network failure occurs, the BIC uses multiplicative decrease in correcting the cwnd

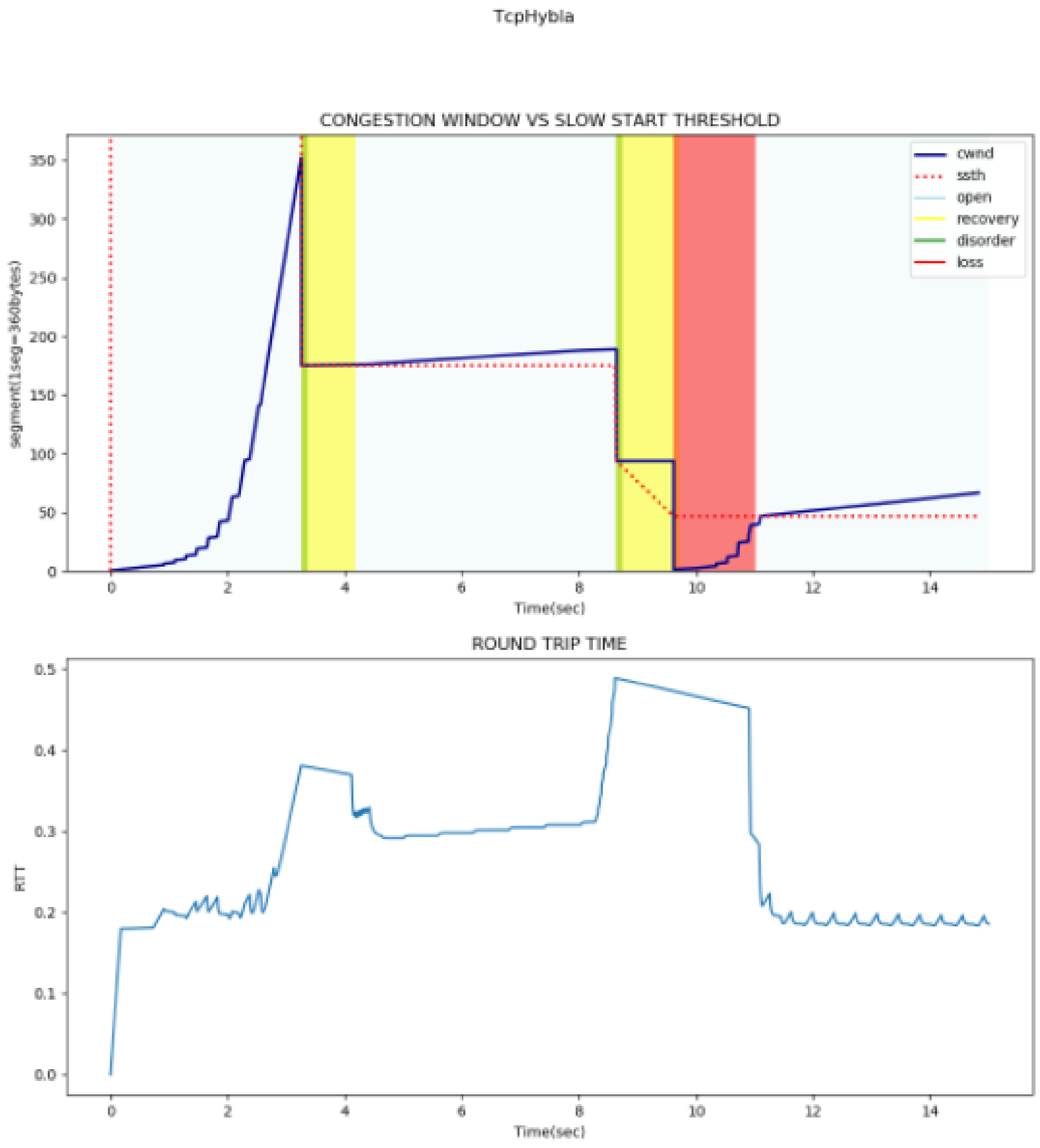
Plot:



➢ Hybla

TCP Hybla aims to eliminate penalties to TCP connections that incorporate a high-latency terrestrial or satellite radio links. Hybla improvements are based on analytical evaluation of the congestion window dynamics.

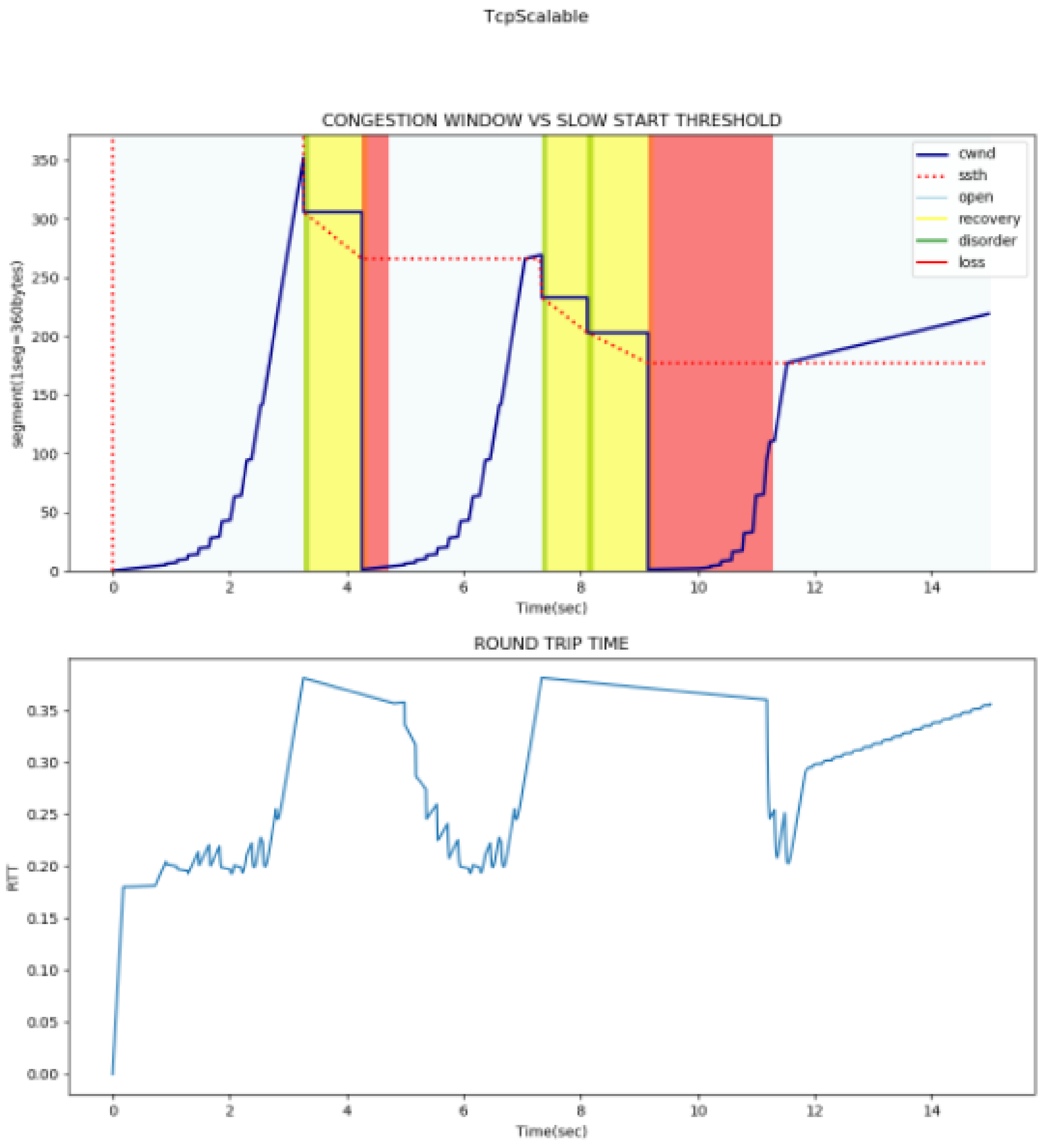
Plot:



➢ Scalable

Instead of halving the congestion window size, each packet loss decreases the congestion window by a small fraction (a factor of 1/8 instead of Standard TCP's 1/2) until packet loss stops. When packet loss stops, the rate is ramped up at a slow fixed rate instead of the Standard TCP rate that's the inverse of the congestion window size.

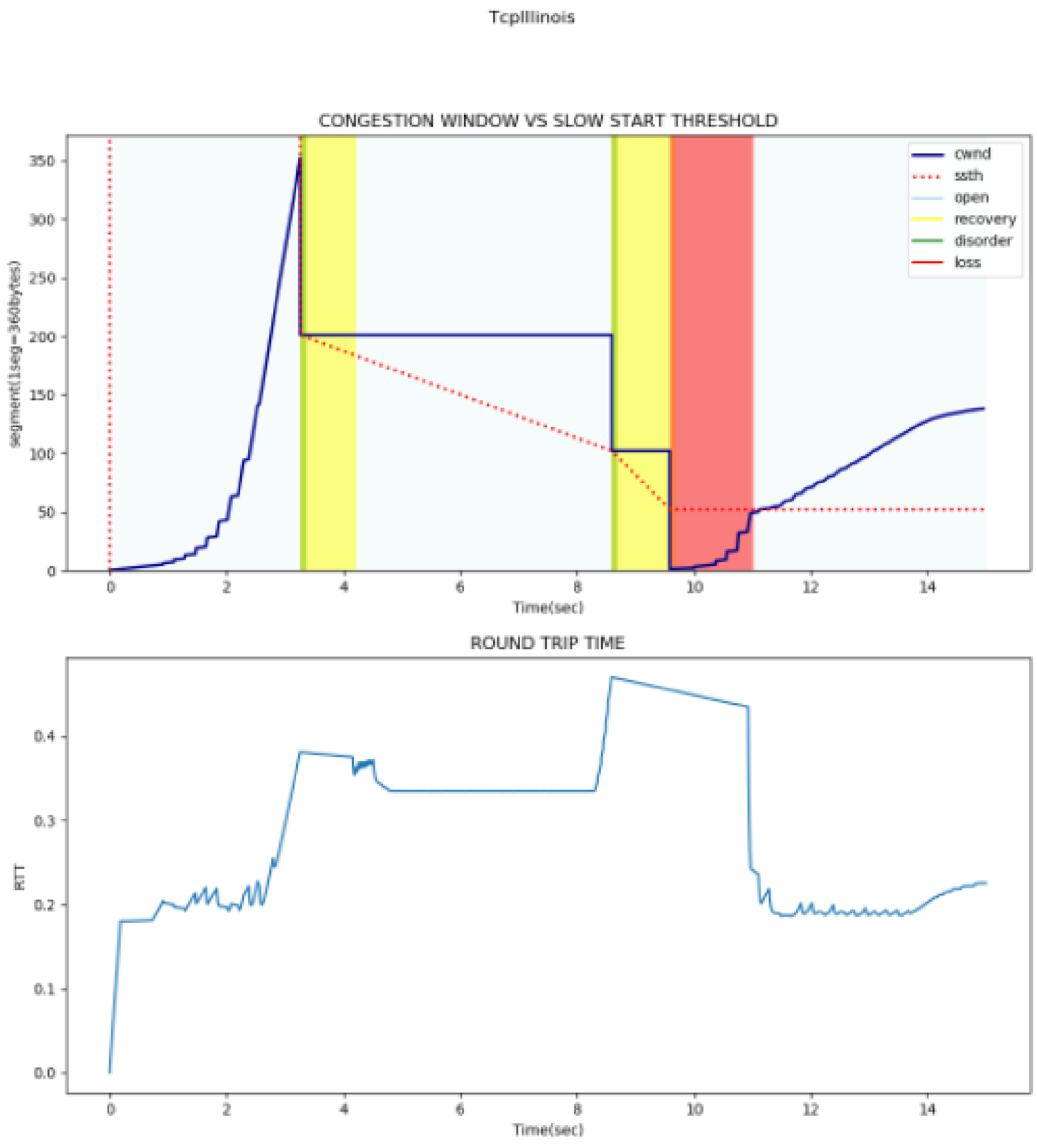
Plot:



- **Illinois**

A sender side modification to the standard TCP congestion control algorithm, it achieves a higher average throughput than the standard TCP, allocates the network resource fairly as the standard TCP, is compatible with the standard TCP. It is a loss-delay based algorithm, which uses packet loss as the primary congestion signal to determine the direction of window size change, and uses queuing delay as the secondary congestion signal to adjust the pace of window size change's-Illinois increases the window size W by alpha/W for each acknowledgment, and decreases W by beta*W for each loss event, where alpha,beta are not constants.

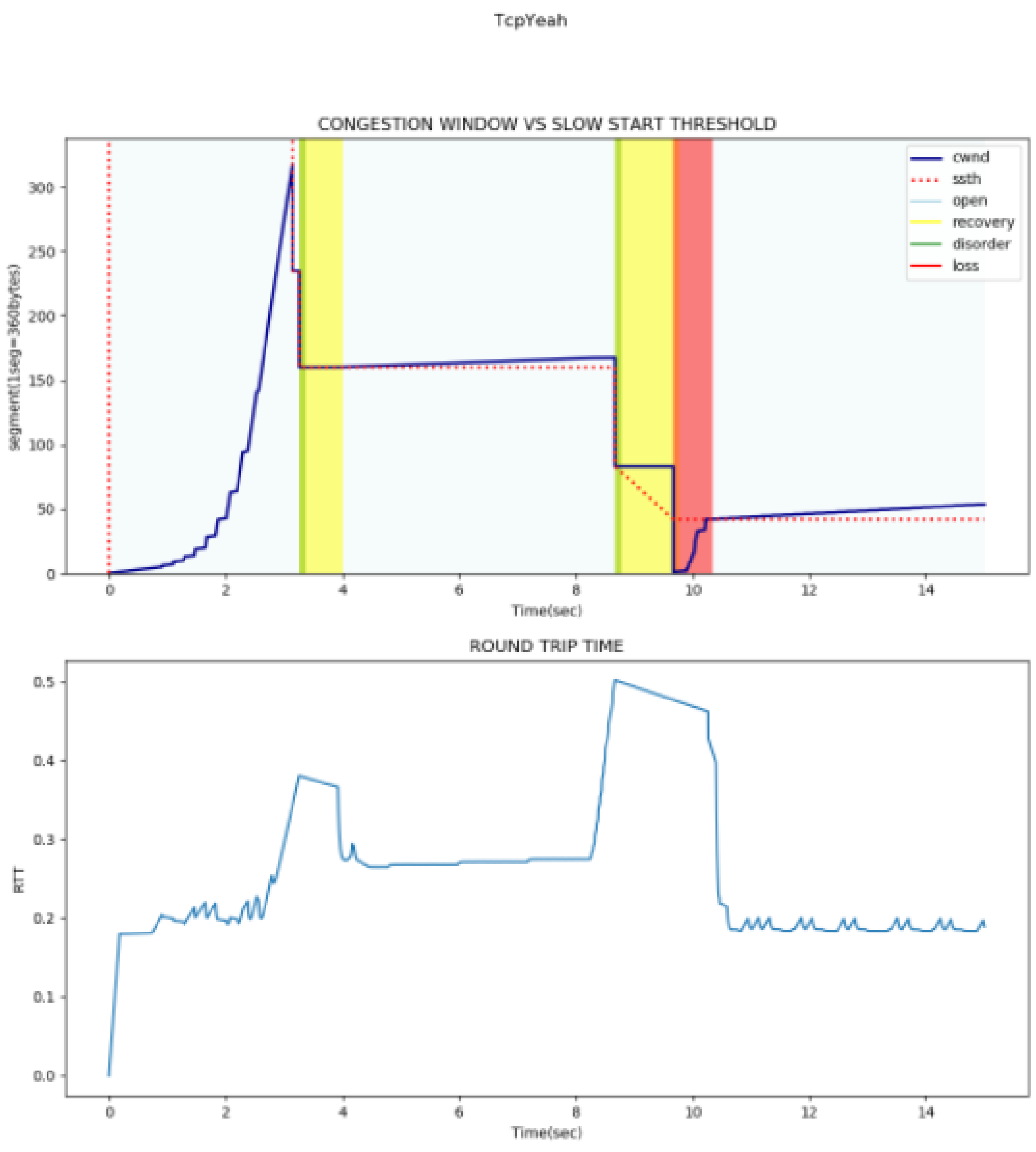
Plot:



➢ Yeah

TCP is a high speed TCP congestion control algorithm which uses a mixed loss/delay approach to calculate congestion windows. Its purpose is to target high efficiency, fairness, and minimizing link loss while keeping network elements load as low as possible. Through delay measures, when the network is sensed unloaded it will quickly exploit the available capacity, trying to keep the network buffer utilization always lower than a threshold

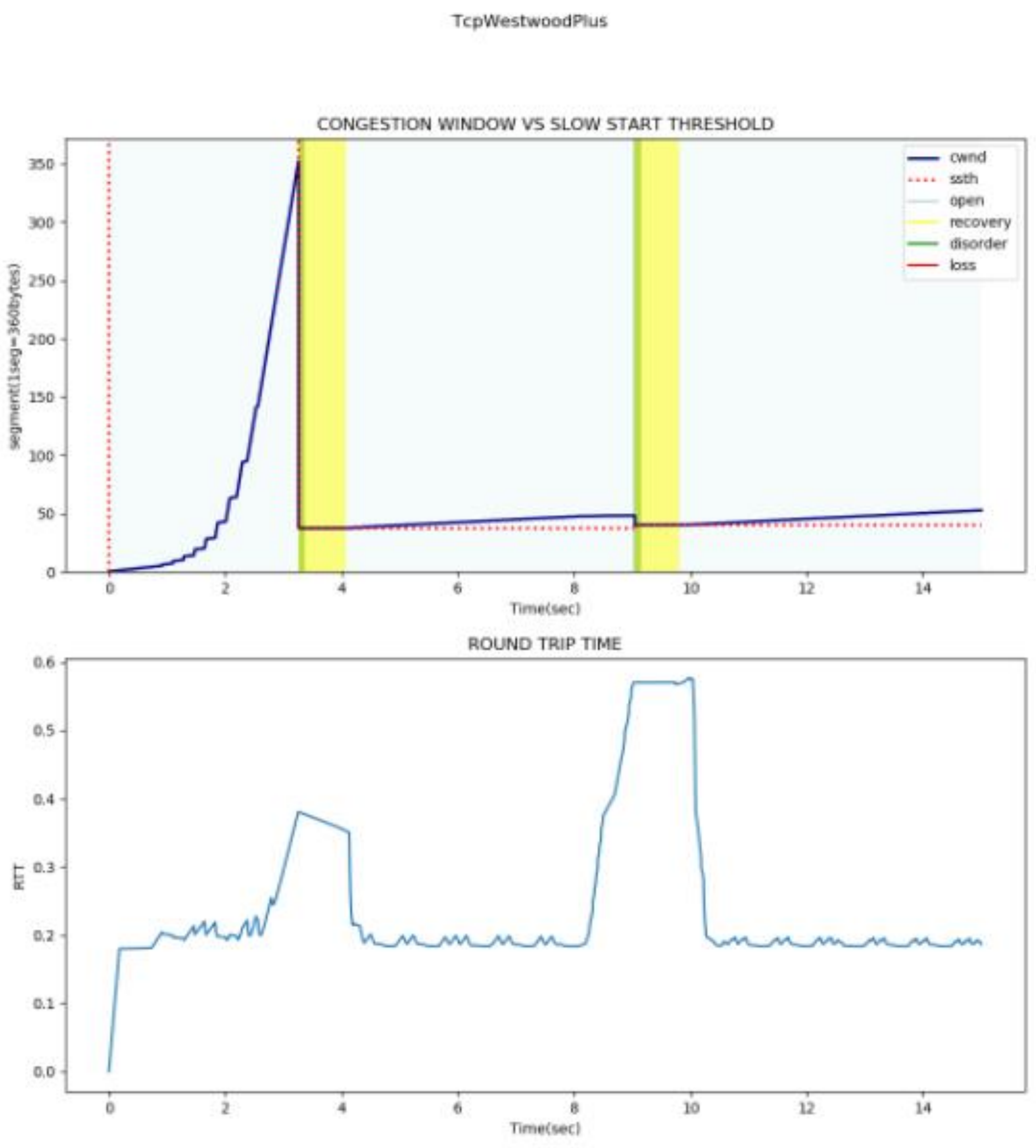
Plot:



➢ Westwood Plus

TCP Westwood+ is an evolution of *TCP Westwood*. TCP Westwood+ is a sender-side only modification of the TCP Reno protocol stack that optimizes the performance of TCP congestion control over both wireline and networks Westwood+ is based on end-to-end bandwidth estimation to set congestion window and slow start threshold after a congestion episode, that is, after three duplicate acknowledgments or a timeout. The bandwidth is estimated by properly low-pass filtering the rate of returning acknowledgment packets

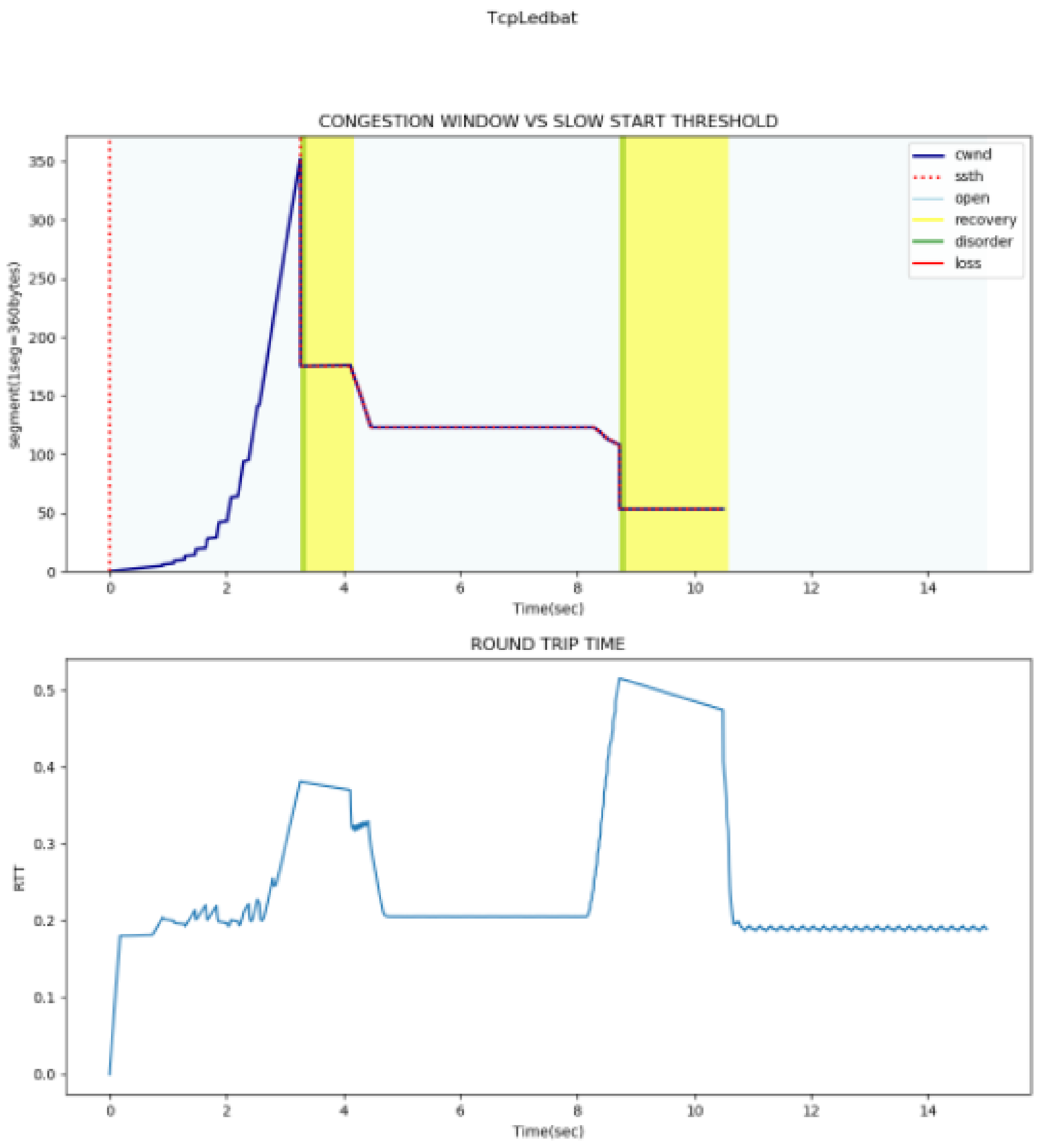
Plot:



- **Tcp Ledbat**

Low Extra Delay Background Transport (LEDBAT) is an experimental delay-based congestion control algorithm that seeks to utilize the available bandwidth on an end-to-end path while limiting the consequent increase in queueing delay on that path.
LEDBAT uses changes in one-way delay measurements to limit congestion that the flow itself induces in the network. LEDBAT is designed for use by background bulk-transfer applications to be no more aggressive than standard TCP congestion control (as specified in RFC 5681) and to yield in the presence of competing flows, thus limiting interference with the network performance of competing flows.

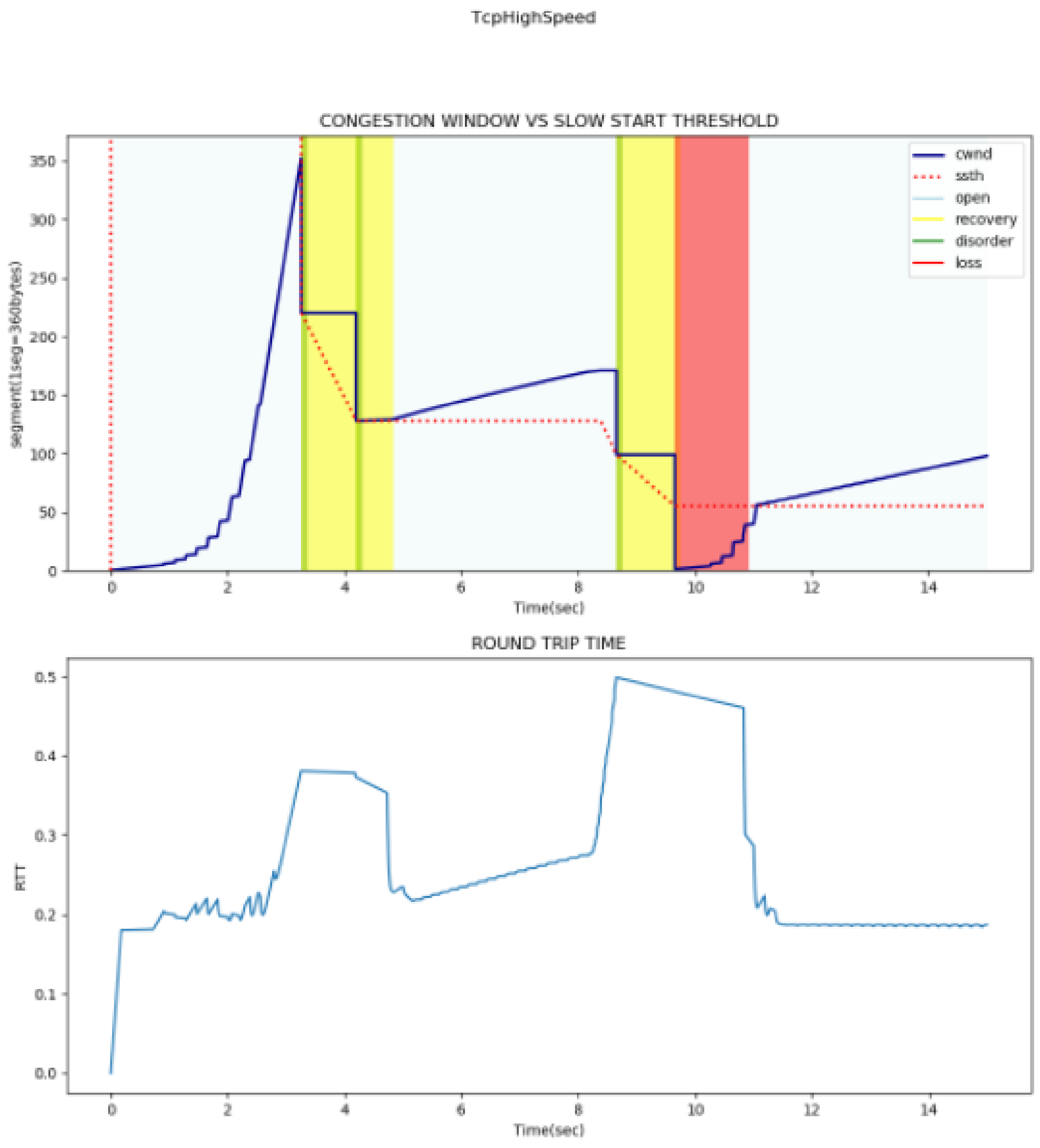
Plot:



➢ Tcp HighSpeed

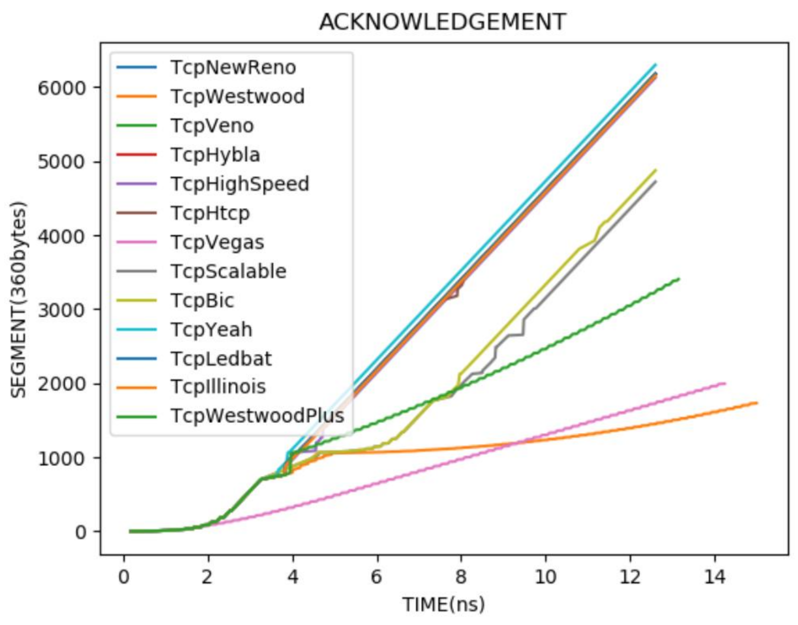
HighSpeed TCP (**HSTCP**) is a congestion control algorithm protocol defined in RFC 3649 for Transport Control Protocol (TCP). Standard TCP performs poorly in networks with a large bandwidth-delay product. It is unable to fully utilize available bandwidth. HSTCP makes minor modifications to standard TCP's congestion control mechanism to overcome this limitation.
When an ACK is received (in congestion avoidance), the window is increased by a(w)/w and when a loss is detected through triple duplicate acknowledgments, the window equals (1-b(w))w, where w is the current window size
This means that HSTCP's window will grow faster than standard TCP and also recover from losses more quickly

Plot:

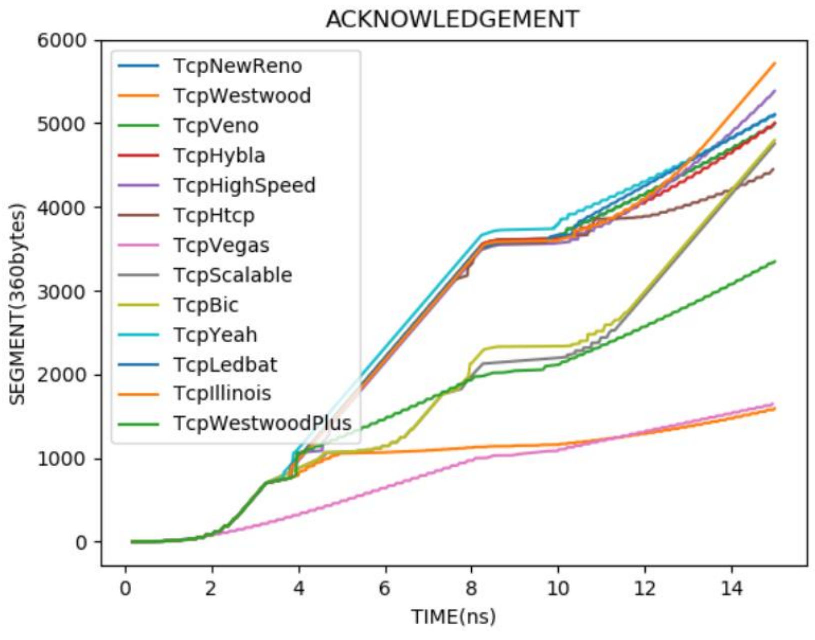


ACKNOWLEDGE VS TIME COMPARISON OF TCP VARIANT WITH VARIANTION IN ERROR MODEL OF NS3

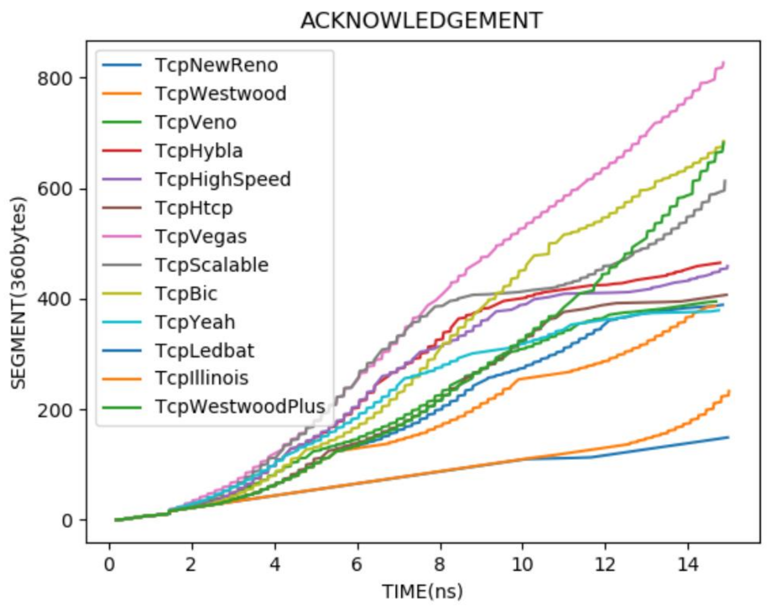
0 CBR(constant bit rate) generator used



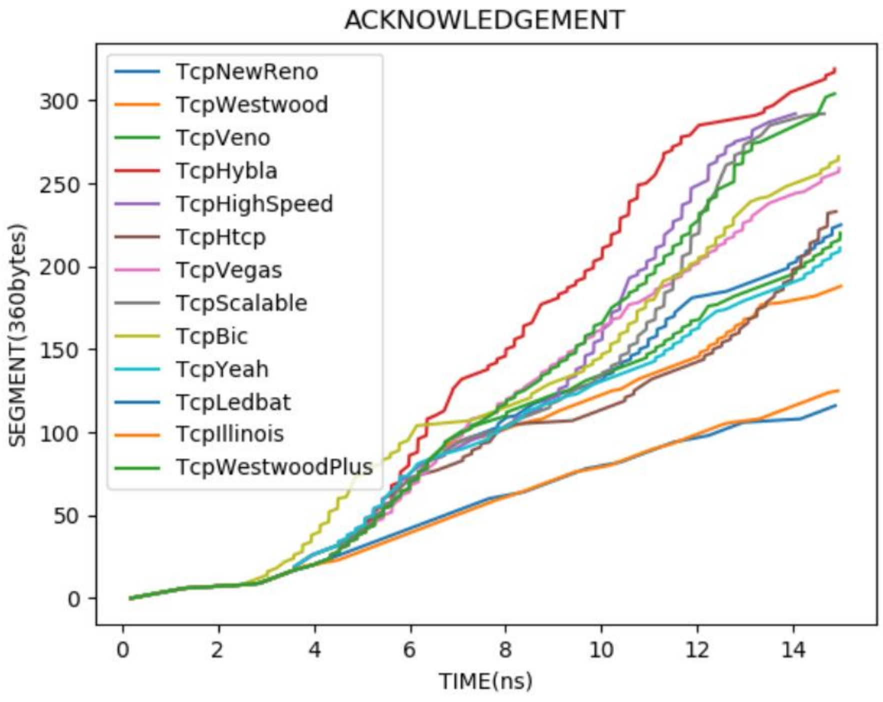
5 CBR(constant bit rate) generator used



At packet error rate=2%



At packet error rate=5%



Github link: https://github.com/Baljotsinghchoudhary/ns3-Tcp-comparison