Experimentation with TCP versions using Mininet

Daniele Romanini
daniele.romanini@aalto.fi

Dmitri Tsumak
dmitri.tsumak@aalto.fi

Manoj Kumar
manoj.kumar@aalto.fi

***Abstract*—Network Scientists have developed various TCP algorithms. These variants have their own pros and cons depending upon the network conditions they are implemented in. Some protocols are good for networks closed to ideal ones, i.e. least delay and packet loss, while modern algorithms are written considering relevant high bandwidth along with significant delay and packet loss. We analyzed the behavior of the three TCP variants called Reno, Vegas and Cubic, with different network conditions. This paper presents the behavior of these protocols without competing with each other for network resources. Conclusions are made on the basis of variation in congestion window, throughput and round trip time under different delay, loss rate and bandwidth.**

Keywords: TCP, Bandwidth, Delay, Packet loss, Congestion, Cognestion window (CWND), Slow Start Threshold (SSTRESH), Throughput (THR), Round trip time (RTT), Mininet,

I. Introduction

Nowadays, the use of TCP has become essential in most of the data transfers. Video streaming websites such as YouTube and Netflix use TCP for serving videos to their users [1]. Paid video streaming websites have to use TCP for better authentication and user validation. Such equipments need already reliable TCP to be very efficient in terms of bandwidth usage and congestion control. Network scientists have been continuously working on improving TCP and have come up with different TCP version. New TCP variants are required to have new efficient features while inheriting all basic properties of standard TCP protocol. Advanced TCP protocols emphasis more on better utilization of bandwidth and early detection of congestion of packet loss than earlier ones, which is well justified considering the network traffic nowadays respect to the past. Different TCP algorithms behave differently in different network conditions. Advanced ones, such as Cubic, give better performance on high bandwidth and high delay scenarios than earlier versions such as Vegas and Reno.

This paper observes how different TCP variants behave in networks with various performance properties. It describes the behavior, advantages and disadvantages of Cubic, Reno and Vegas algorithms in networks with different bandwidth, packet loss and delay properties. The comparison is made on the basis of parameters like congestion window/slow start threshold, throughput and round trip time.

The choice of various scenarios properties was made in order to observe the features of each algorithms (i.e. Cubic is made for high bandwidth and high delay scenarios). The choice is justified in each scenarios description (section 4).

II. Background

As quoted in [2] "TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications." TCP provides Reliability, flow control and multiplexing along with basic data transfer between two interconnected hosts. For reliability, TCP ensures that any lost, damaged or duplicated data should be recovered by using the concept of acknowledgement and sequence numbers. TCP uses a congestion window to achieve flow control between sender and receiver. Basic TCP protocols use the concept of slow start and congestion control [3].

Using slow start, TCP observes the rate at which packets are required in the medium of the rate of received acknowledgments. Slow starts initializes the sender's congestion window at some initial value (i.e.: 1 or 2 segment size) at the starting of a new connection and keep increasing it according to the number of acknowledgement received from receiver's end. Once congestion event occurs, TCP either decreases CWND to a certain point or start from beginning with a slow start depending upon the TCP variant. These are the fundamentals of TCP and being used in all TCP versions.

III. Experimentation setup

A. Environment

In order to test algorithms in different life scenarios, we used `Mininet` emulation platform. We run it on an isolated virtual machine, so that host machine would not be able to influence experiments outcome with its own traffic. The guest machine operating system was `Ubuntu 16.04` minimal and it was already preconfigured for Mininet usage by a Mininet team.

B. Tools

For traffic generation we used `iperf3` tool. The advantages of this tool is that it allows to separate hosts on servers and clients. In addition, it allows to set what congestion control algorithm will be used, time for data exchange, intervals between transmissions, etc.

In experimentations following `iperf3` commands were used on Mininet hosts:

Listing 1. iperf3 server

```
$ iperf3 -s
```

Listing 2. iperf3 client and data transmission for 60 seconds

```
$ iperf3 -c 10.0.0.2 -i 1 -C cubic -t 60
```

For monitoring and capturing traffic we used TCPdump and TCPprobe. In experimentations following TCPprobe commands were used on the virtual machine (not in Mininet environment) to start capturing packets information to TCP_probe.txt file from Mininet network:

Listing 3. Enable TCPprobe and and capture traffic information to a file

```
$ rmmod TCP_probe >/dev/null 2>&1
$ modprobe TCP_probe port=5201 full=1
$ cat /proc/net/TCPprobe > TCP_probe.txt &
```

To capture TCPdump data we run the following commands:

Listing 4. Capturing network traffic information with TCPdump

```
$ TCPdump -w ./client.log # on a client host
$ TCPdump -w ./server.log # on a server host
```

For plotting we used TCPtrace, xplot.org and gnuplot tools. In order to plot network throughput captured with TCPdump, TCPtrace in combination with xplot.org was used:

Listing 5. Plot graph for network throughput

```
$ TCPtrace T TCPdump.log
# a2b_tput.xpl will be generated
# by TCPtrace
$ xplot.org -x a2b_tput.xpl
```

To plot data about round-trip time the following command was used:

Listing 6. Plot graph about round trip time

```
$ TCPtrace R TCPdump.log
# a2b_rtt.xpl will be generated
# by TCPtrace
$ xplot.org -x a2b_rtt.xpl
```

To plot congestion window size and slow-start threshold captured with TCPprobe, we used the following script:

Listing 7. Plot graph about cwnd and SSTHRESH

```
#! /bin/bash
gnuplot -persist <<EOF
set terminal png size 1500,1000 \
 enhanced font 'Times,20'
set output 'graphTCP.png'
# blue
set style line 1 lc rgb '#0060ad' \
 lt 1 lw 2 pt 5
# red
set style line 2 lc rgb '#dd181f' \
 lt 1 lw 2 pt 7
set style data linespoints
set title "$1"
# this parameter should be optimized
# according to the graphs scale
set yrange [140:145]
# this parameter should be optimized
# according to the graphs scale
set xrange [45:55]
set xlabel "time(seconds)"
set ylabel "Segments (cwnd, SSTHRESH)"
plot "$1" using 1:7 title "snd_cwnd", "$1" \
 using 1:(\$8>=2147483647? 0: \$8) \
 title "snd_sstresh"
EOF
```

C. Network topology

We used Mininet's default network topology for the experiments. It is described in a following figure:

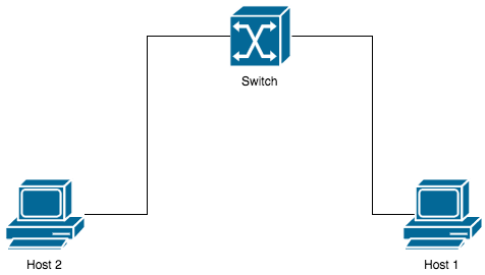

Fig. 1. Simulation results for the network.

We used following command to start mininet network topology with specified bandwidth, delay and packet loss:

Listing 8. Plot graph about cwnd and SSTHRESH

```
$ mn --link tc,bw=val1,delay=val2ms,loss=val3
```

where val1,val2, val3 are substituted according to different scenarios.

IV. Experiments

Some generated graphs are attached at the end of this document. If you want to see all the graphs, it is possible to see or download them from this link: goo.gl/mnr2St

A. Scenario 0

Bandwidth (Mbps)	Latency (ms)	Packet loss (%)
10	0	0

This is an ideal scenario (no delay, no loss). The purpose of running the experiment under this settings was to see the behavior of all three algorithm on an ideal network. The THR graph is almost the same for all the three algorithm.
Looking at the CWND and SSTRESH graphs, we can see that the Congestion Window of Reno increases exponentially until the limit (this limit is the threshold value of the receiver). The congestion window of Cubic changes accordingly to a Cubic function; the SSTRESH is increased after some time (some ACK have been received) and the CWND raises accordingly. Conversely, Vegas behaves differently: there is an exponential growth at the beginning, but then we can see that the CWND goes down at some points, for increasing again later. The general trend of the curve is to go down: this is due to the fact

that Vegas increases or decreases the congestion window size based on the network utilization. Scenario 0: Note: Scenario with Bandwidt = 200 Mbps, Delay = 0 ms, Loss = 0% was not considered as ideal, because the trend of the curves (i.e.: CWND, SSTRESH) would have been the same (since delay and loss are null) independently to the bandwidth.

B. Scenario 1

Bandwidth (Mbps)	Latency (ms)	Packet loss (%)
200	10	2

Looking at the properties of this scenario (High Bandwidth, Low delay and Low Loss rate) the algorithms are expected to perform well. In fact, the THR are among the highest ones in all the scenarios. In this case, the worst value is the Vegas' one (around 150000 in comparison to more than 175000 of Cubic and Reno). Looking at the RTT graphs, Vegas seems also to be the slowest one, and the one experiencing more packet loss (when RTT increases, a packet loss has probably occurred, because the sender was waiting for an acknowledge for a long time). This can be observed also in the CWND and SSHTRESH graphs, where Vegas experiences more fluctuations respect to the other two algorithms (Vegas detects the congestion based on RTT).
At the beginning, all the algorithms increase their congestion window dramatically their congestion window: at the first congestion event, Reno and Vegas put the Congestion Window as an half of the value reached before the event, meanwhile Cubic decreases it accordingly to its cubic function. It is easy to notice that the maximum value reached by the CWND in Cubic is much smaller than the one of Vegas and Reno. Using Cubic, due to packet loss and timeouts the window and threshold decreased and the curves stabilised at around 6 segments for CWND and 4 segments for SSHTRESH.
In spite the fact that Reno average window size and threshold are quite similar to Cubic one (6 segments for cwnd and 4 segments for SSTHRESH), its values increased and decreased more frequently. The reason for that could be the way with Cubic algorithm recover after packet loss or timeout. Cubic algorithm tries to be as closer as possible to Wmax value, so the number of transmitted packets will be more steady. Reno, however, (everytime that a congestion events occurs), it does not wait until the timeout to detect loss, but uses fast retrasmist (1), setting CWND at half of its value, and then the SSHTRESH , which results in more scattered graph. The reason for this (more fluactions of the CWND curve in the same time frame) is that Reno Fast Recovery ensures that channel does not remain empty at the time of congestion control. (2)

1) In the fast retransmit, instead of waiting till time out to detect loss, Reno used duplicate acknowledgements. It waits till three acknowledgments before retransmission of the packet. As soon as it knows about the congestion it cut the congestion window by half.
2) In case that Reno keeps getting duplicate acknowledgements after the first retransmission, it assumes the receiver is removing the packets from medium and it keep filling the medium with new packets until it gets the acknowledgement for retransmitting packet. Once it get the acknowledgment of retransmitted packet it again invoke the congestion avoidance by applying linear increase in CWND window.

C. Scenario 2

Bandwidth (Mbps)	Latency (ms)	Packet loss (%)
200	100	5

This scenario has been set for observing the features of Cubic in a scenario it was designed for (high bandwidth, high delay). Cubic, in fact, is performing much better than Reno: The THR is considerably higher: it is notable from the difference of scale in the THR graphs (Cubic THR is a bit less than 50000 bytes/s, meanwhile Reno's one is around 1000 bytes/s). Moreover, for Reno, TCP Probe was not able to collect any data (other signal about the very low throughput of Reno). RTT for Cubic is more linear and it does not vary too much over the all capture phase (apart from one time in the middle, where it dramatically increases, but this is probably due to a packet loss event).
After the initial exponential growth of CWND, the behaviour of the CWND is very repetitive: when loss events happen, Cubic records the window size as `W_Max`, and start to decrease it. The value of CWND is supposed to be stable around `W_Max`: it continues to decrease and increase again (to `W_Max`) because of the quiet high loss of that scenario. Its behaviour is anyway less aggressive than other standard TCP algorithms. In fact, Cubic uses convex and concave profiles to adjust window sizes, which is independent from RTT (this is why it perform well in high delay networks like this one).
The behaviour of Cubic can also be seen in Scenario 4, with less packet loss rate. Vegas THR is as good as Cubic. The theoretical behaviour of Vegas can be noticed in the CWNDS and SSTRESH graph: it is not continuing increasing the congestion window during congestion avoidance, but it checks congestion by measuring current throughput and comparing it to the expected one, and increases congestion window only if these values are similar. Vegas performs a packet retransmission whenever the RTT exceeds the timeout value. This is why, in this scenario, the congestion window is not dramatically dropping to a very low value often. It is also possible to see that, whenever a packet is retransmit, the CWND drop to the value of the SSHTRESH most of the times. The SSHTRESH value increases every 2 acknowledged packets. In case of single packet loss, congestion window is reduced by only 1/4 (i.e: from 4 to 3, as can be noticed in the right part of the graph).

D. Scenario 3

Bandwidth (Mbps)	Latency (ms)	Packet loss (%)
10	100	5

This is another scenario that differs from the above (Scenario 2) just for bandwidth. The other path properties (Delay, Loss) are exactly the same. It just confirms that Reno is not

designed for high delay and high loss scenario. In fact we do not get any TCP probe Reno's data also for this one. (It is actually feasible: if there were no data collected with exactly the same path properties with higher bandwidth, there should not be for that one as well). The behaviour of the other protocols is quite similar to Scenario 2, but with lower performance (e.g.: considerably lower THP, less than 10000 for Cubic less than 20000 for Vegas, in comparison to 50000 of Scenario 2).

E. Scenario 4

Bandwidth (Mbps)	Latency (ms)	Packet loss (%)
200	100	2

Theoretically, in this scenario it would be possible to see the good behaviour of Cubic and Vegas. In fact, the path properties of this scenario are the same of Scenario 2, apart from the packet loss, which is lower in respect to second one (where loss rate is 5 %): as it has been mentioned earlier, Cubic should behave very well in this kind of scenario, and also better compared to Scenario 2 (because of the lower packet loss). Since Vegas behaves good in scenario 2, even better performance is expected from this scenario.
The better behaviour of Cubic in respect to Scenario 2 can be noticed since the Congestion Window increase by 5-6 segments, much more than the average growth in Scenario 2 (2 segments) before decreasing again. Noticeable it is the fact that, after a single packet loss, the value of CWND is set to the current SSTRESH one, while, when a multiple packet loss is experienced, SSHTRESH value decreases as well. The trend of the cubic function can be noticed around 82 seconds in the graphs: the cubic function is composed by two different way of growing, a first concave portion, where the window size raises quickly (up to the dimension reached at the last congestion event, which occurred at 81 s in this case), and a second convex part, increasing slowly before and then more rapidly. Vegas design focus is more on packet delay rather than packet loss and, in fact, the THR in this scenario is much better than the one in Scenario 2 (about 90000 vs 40000 bytes/s). Under these settings, Vegas THR is the best one in comparison (Cubic: almost 65000; Reno: a bit more than 20000) to the other algorithms.
Vegas applies a modified slow start: the congestion window increases every round trip time. In the graph it is possible to notice time frames (i.e: 43 experienced minor changes in large part of the graphs. The less aggressive behaviour of Vegas respect to Reno is also shown: looking at the representation of all the simulation, Vegas' CWND seems to increase and decrease regularly, meanwhile Reno's one experiences much more variance, similarly to scenario 1 from this viewpoint.
In this scenario with high delay, Reno THR is the worst: 20000 bytes/second in comparison to 200000 bytes/second of scenario 1. This means that Reno is considerably affected by the delay.

F. Scenario 5

Bandwidth (Mbps)	Latency (ms)	Packet loss (%)
10	10	2

This scenario is similar to Scenario 1, apart from the considerably lower bandwidth (10 Mbps vs 200 Mbps). Thus, an higher number of congestion events is expected respect to Scenario 1.
In terms of THR, in fact, Reno and Cubic presents lower results: Cubic is anyway performing better than Reno (in contrast with Scenario 1, where Reno experienced the best THR, around 200000). The difference between these two algorithms is anyway not so high. The variation of the Congestion Window seems to be similar, but there is a difference that can be noticed: before a packet loss or a time out, Cubic tryes to mantain the Congestion Window size at his max value (`W_Max`). It is easy to see that in Reno the Congestion Window size drops immediately down, meanwhile cubic maintain for at least another packet the value reached (most of the cases it is just one packet probably because this scenario has low bandwidth, it is unable to send more at one time). Surprisingly, Vegas behaves much more better showing an higher THR. It shows a more aggressive behaviour (with more fluctuantions in the graphs) in comparison to Cubic and Reno, but the it reach an higher value of Congestion Window (exluding the initial part) respect to the other: 26 versus 16. This means that it is better managing network with lower bandwidth.

V. Conclusion

This paper has observed that Cubic responds better at the time of congestion and after congestion in comparison to older variants such as Reno and Vegas. We conduct the experiments on a basic network scenario with two host transferring data through a network switch. However, our main focus was to analyze the behavior of three TCP variants (Reno, Vegas and Cubic) individually in different network conditions as Bandwidth, Delay and Loss. This paper does not observe the behavior of these protocols when they are competing with each other as we are running protocols individually. From all the scenarios, it is easy to notice that Vegas give poorer performance in term of round trip times. All three variants give almost same throughput with minor difference in zero delay and loss scenarios (Scenario 0). However, it is evident the different trend of congestion window for Vegas as it uses RTT to determine congestion instead of packet loss. In high bandwidth-loss scenarios (Scenario 2), Reno shows poorer behavior as it is unable to deal with increased delay and packet loss. We can conclude that with low bandwidth and low delay paths (Scenario 5) Reno and Cubic give almost similar performances which are better than Vegas in terms of RTT.
High delay scenarios are the worst for Reno (looking at the fact that TCP Probe was not able to collect any data in these scenarios, 2 and 3).Vegas in newer than Reno, and correct the issue of congestive packet loss by monitoring RTT and bandwidth. In fact, Vegas perform much better in high delay scenarios (2 or 3). Vegas demonstrates a better management of network resources when low bandwidth is present: Scenario 5, in fact, show that Vegas is the algorithm with high THR, conversely than we could expect (in Scenario 2, in fact, where path properties where the same apart from an higher

bandwidth, Vegas THR was the worst one).
Finally, Cubic is the newest algorithm considered, and it is more advanced since it uses a cubic function to increase and decrease CWND window. This makes Cubic suitable for long fat pipes, but not with low bandwidth.

REFERENCES

[1] Jim Martin, Yunhui Fu, Nicholas Wourms, Terry Shaw: Characterizing Netflix Bandwidth Consumption
[2] Defense Advanced Research Projects Agency, Information Processing Techniques Office: TRANSMISSION CONTROL PROTOCOL
[3] W. Stevens: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms
[4] Hasan Mahmood Aminul Islam: Analysis of RTCWeb Data Channel Transport Options, pp 13-17, 2012
[5] S. Fahmy, M. Sharma: Deployment Consideration for the TCP Vegas Congestion Control Algorithm, 2003
[6] S. Ha, I. Rhee, L. Xu, CUBIC: A New TCP-Friendly High-Speed TCP Variant, 2008
[7] Matti Siekkinen: CS-E4130 Computer Networks II - Course Slides, TCP/IP Advanced
[8] The Linux foundation Wiki: tcp_testing
[9] Mininet: An Instant Virtual Network on your Laptop.

List of figures

Scenario 0

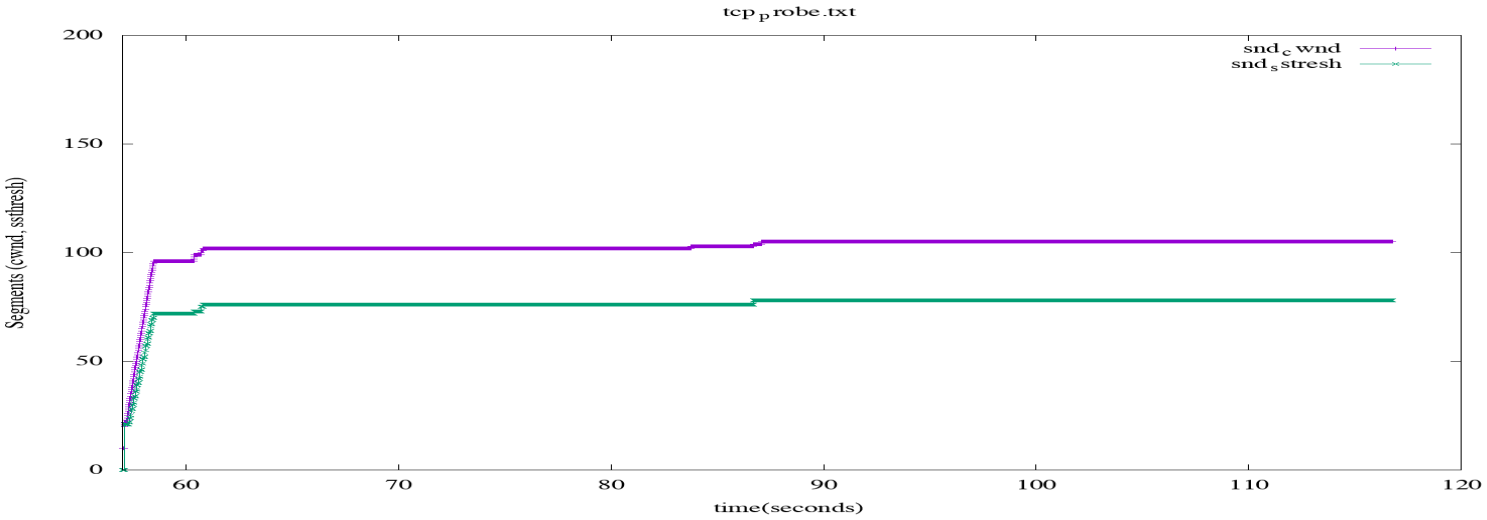

Figure 1: Cubic CWND/SSHTRESH scenario 0

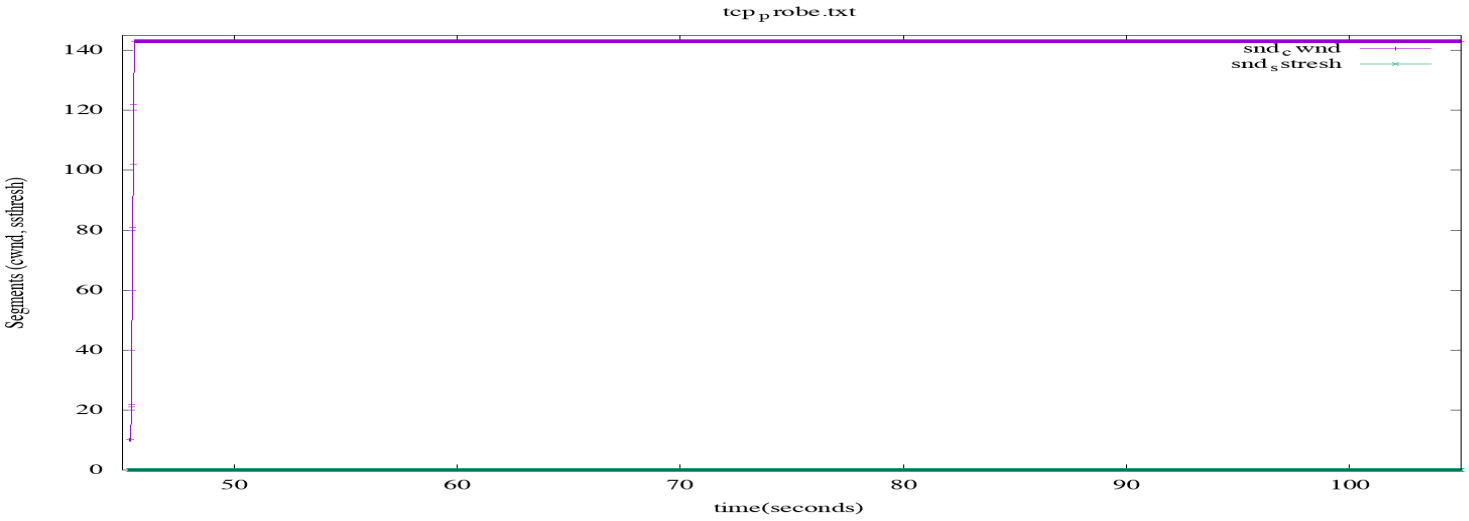

Figure 2: Reno CWND/SSHTRESH scenario 0

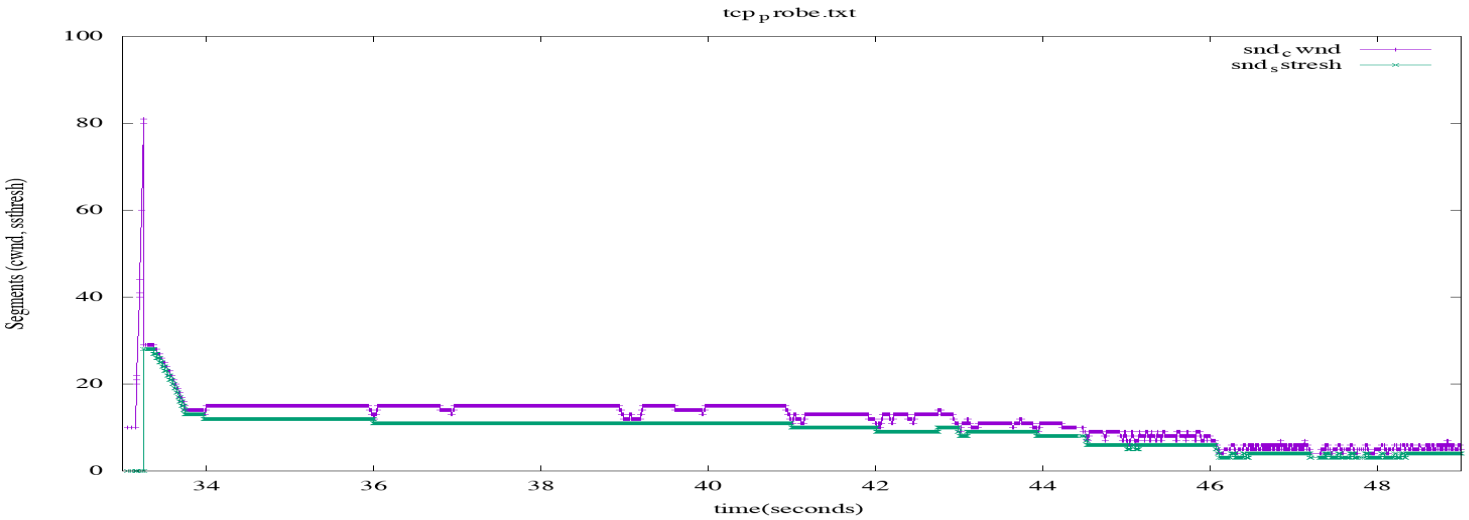

Figure 3: Vegas CWND/SSHTRESH scenario 0

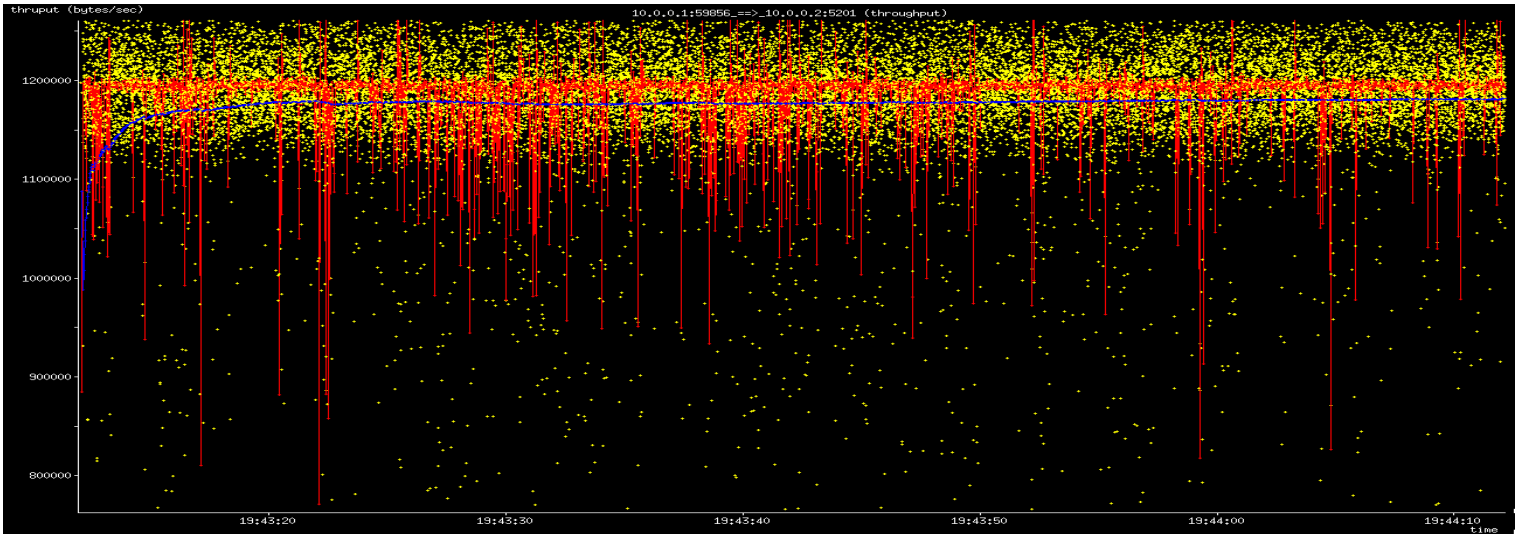

Figure 4: Cubic THR scenario 0

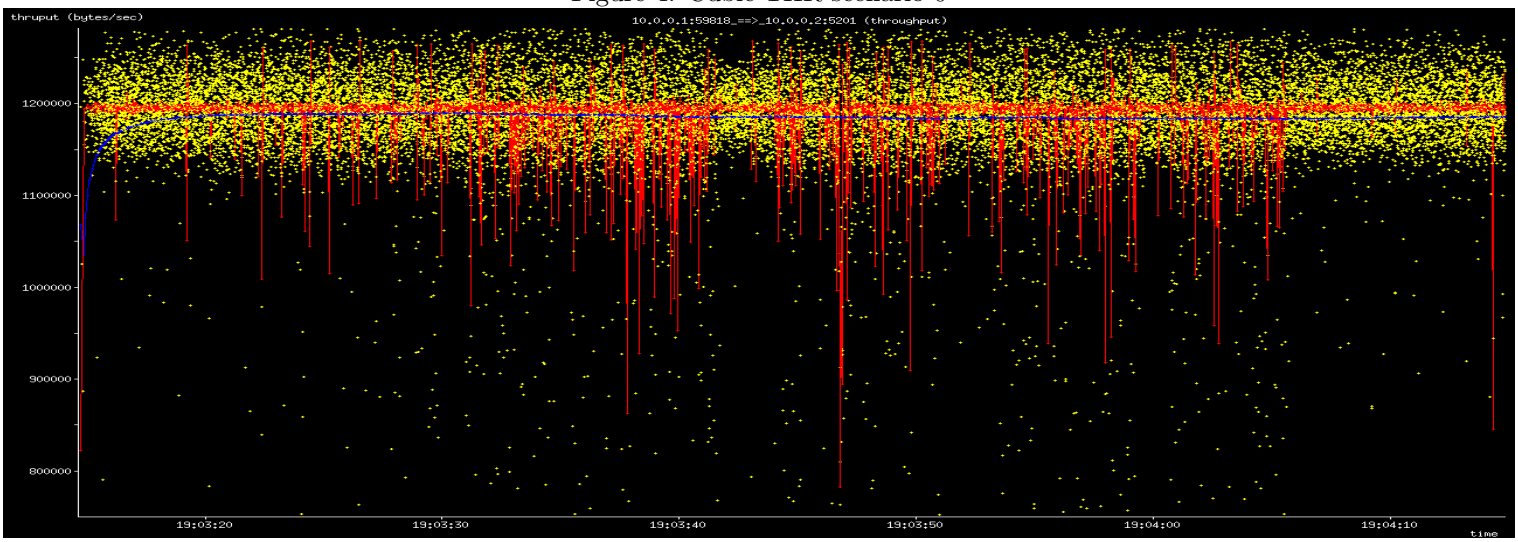

Figure 5: Reno THR scenario 0

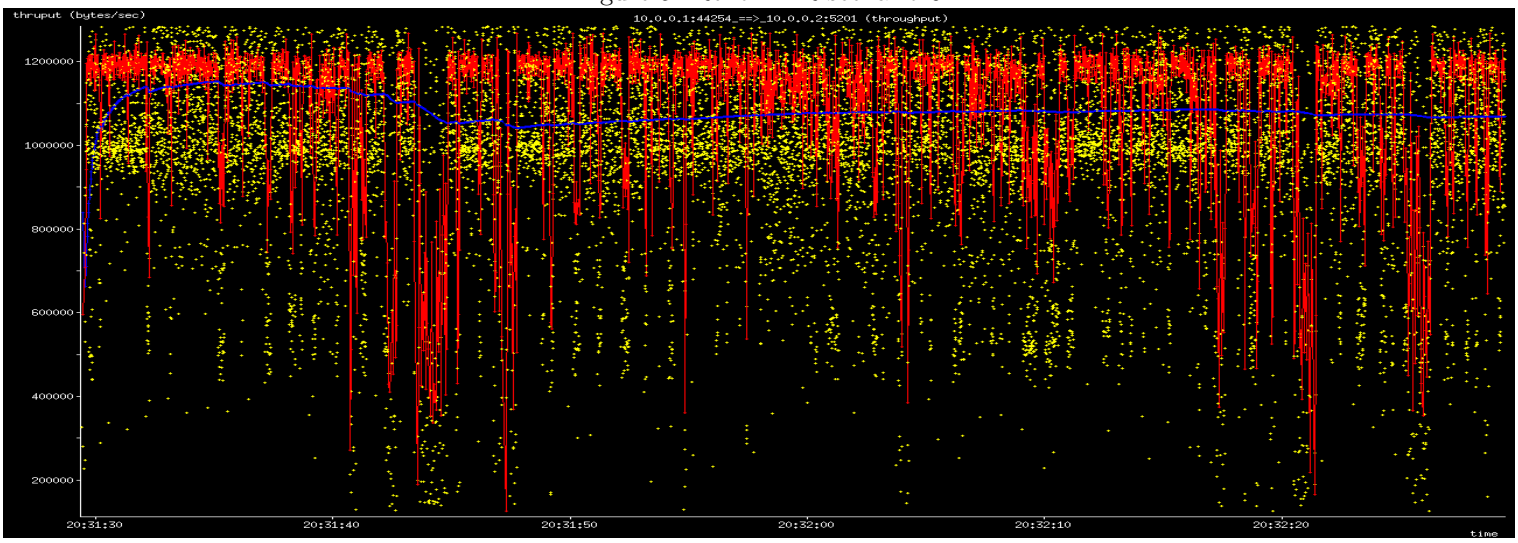

Figure 6: Vegas THR scenario 0

Scenario 1

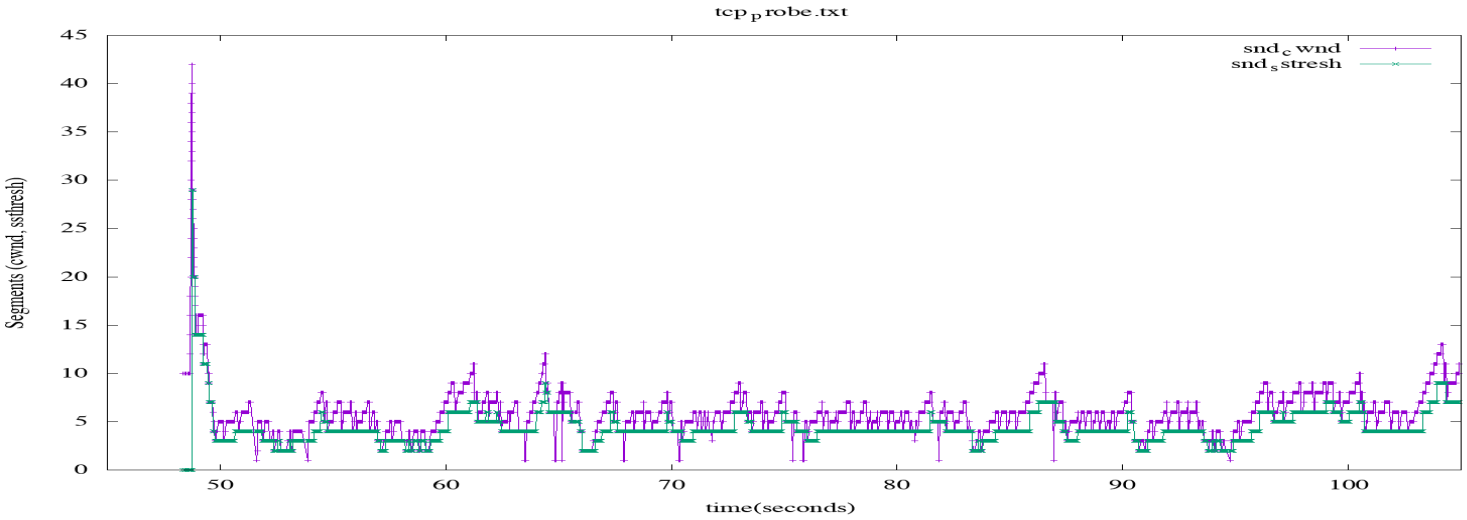

Figure 7: Cubic CWND/SSHTRESH sc. 1

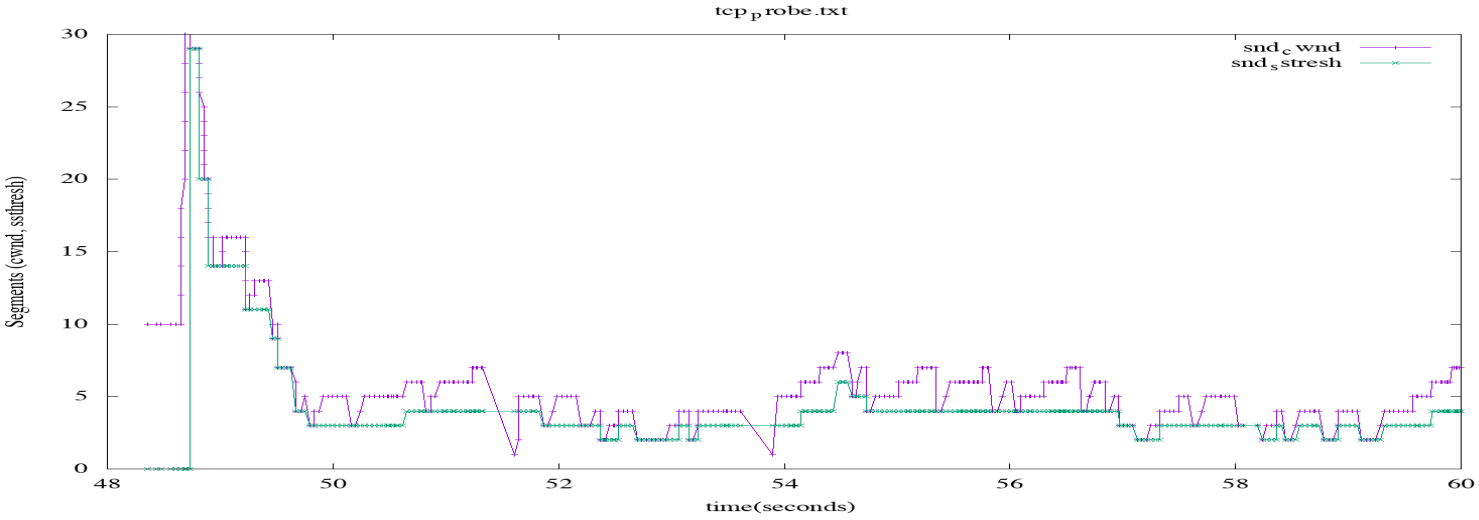

Figure 8: Cubic CWND/SSHTRESH zoom sc. 1

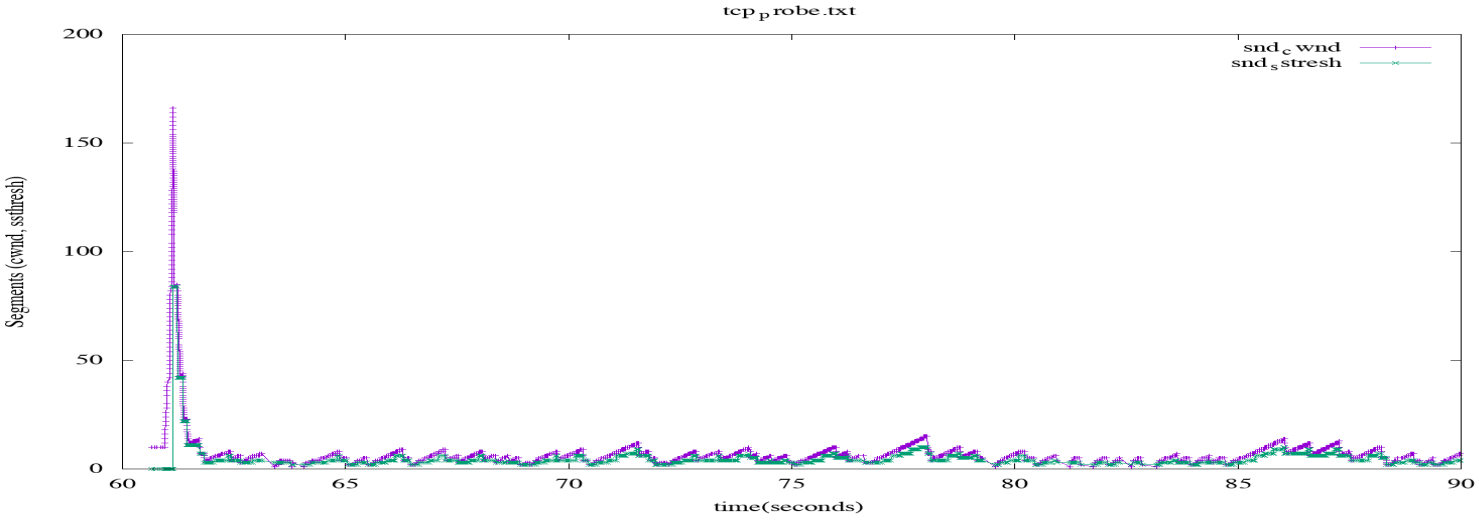

Figure 9: Reno CWND/SSHTRESH sc. 1

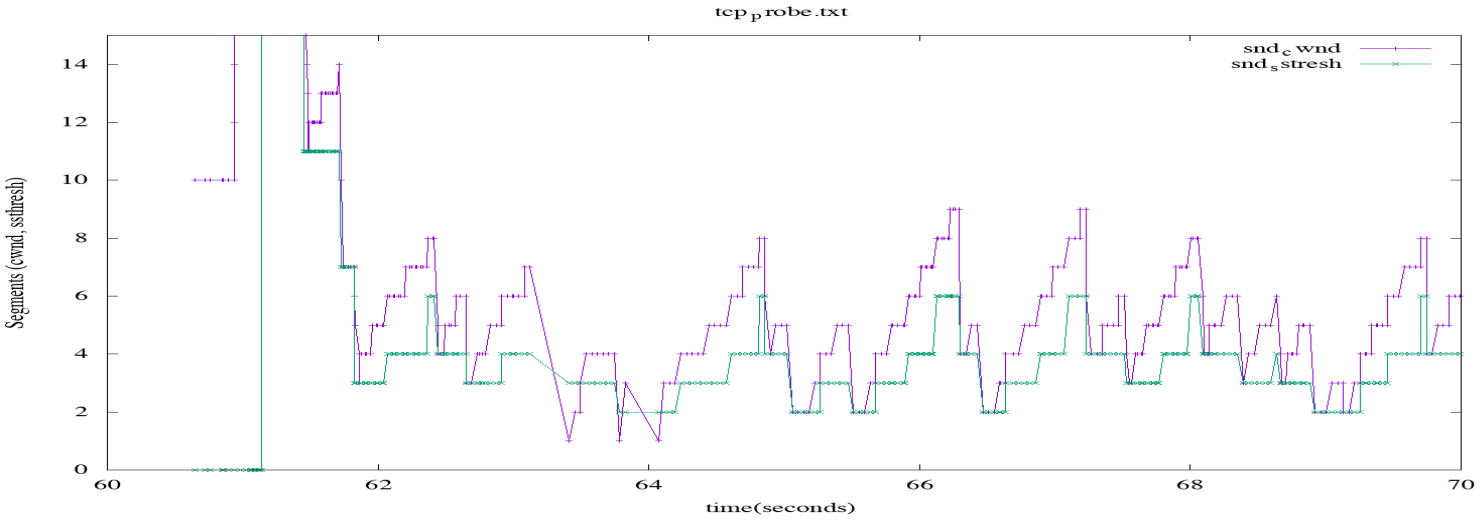

Figure 10: Reno CWND/SSHTRESH zoom sc. 1

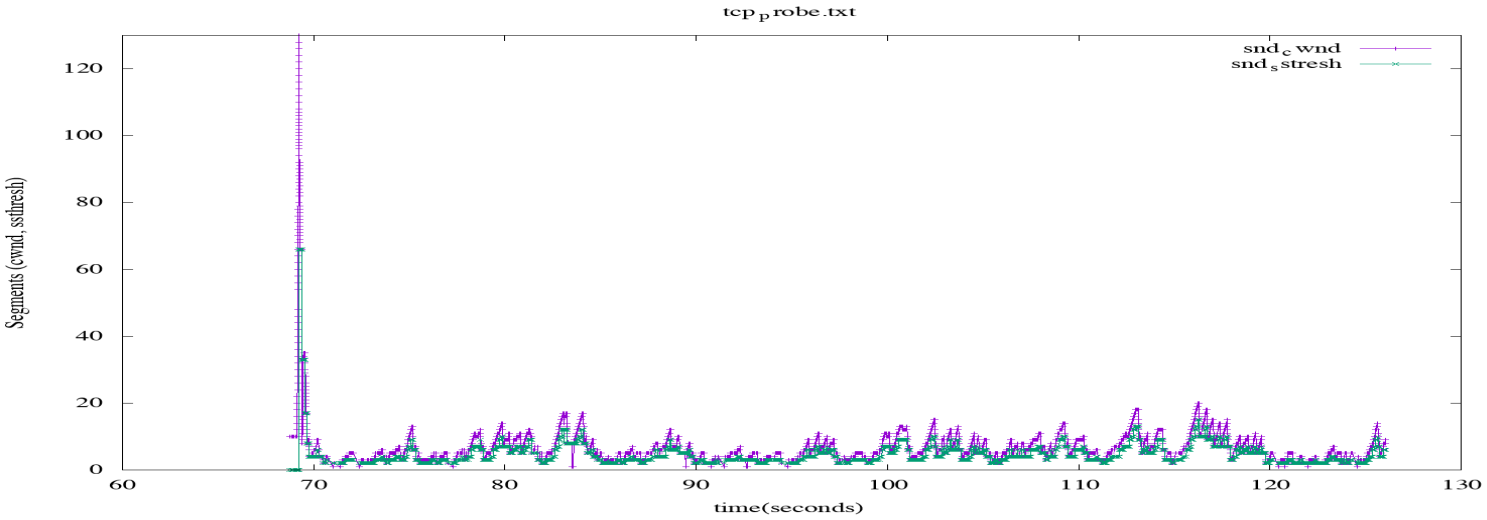

Figure 11: Vegas CWND/SSHTRESH sc. 1

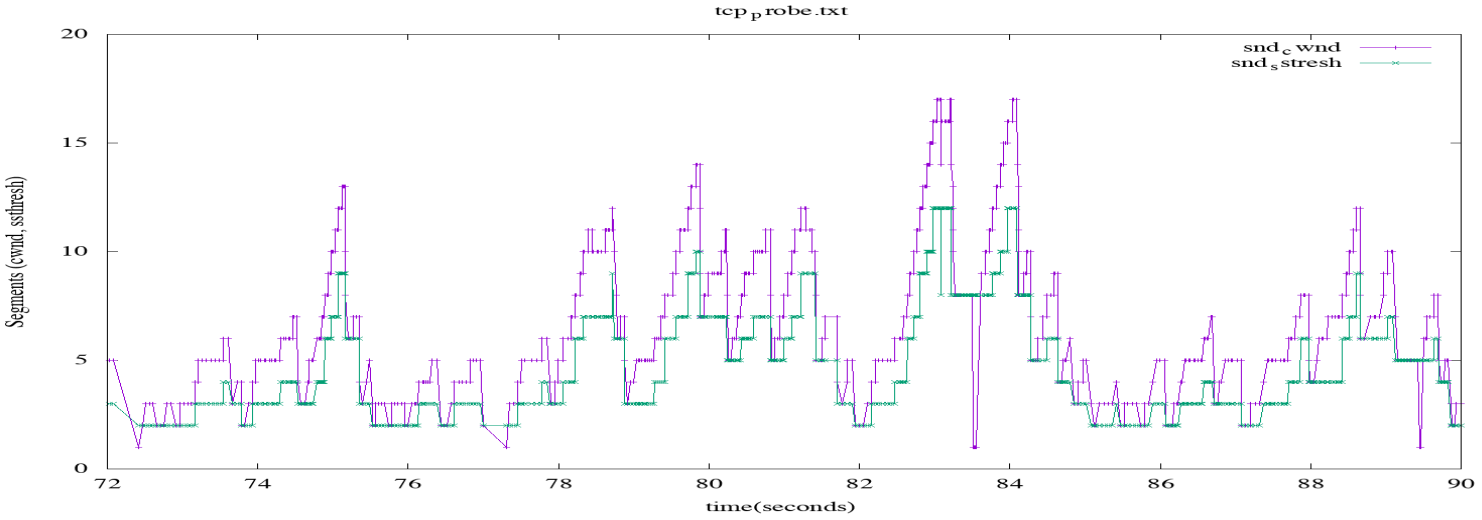

Figure 12: Vegas CWND/SSHTRESH zoom sc. 1

Scenario 2

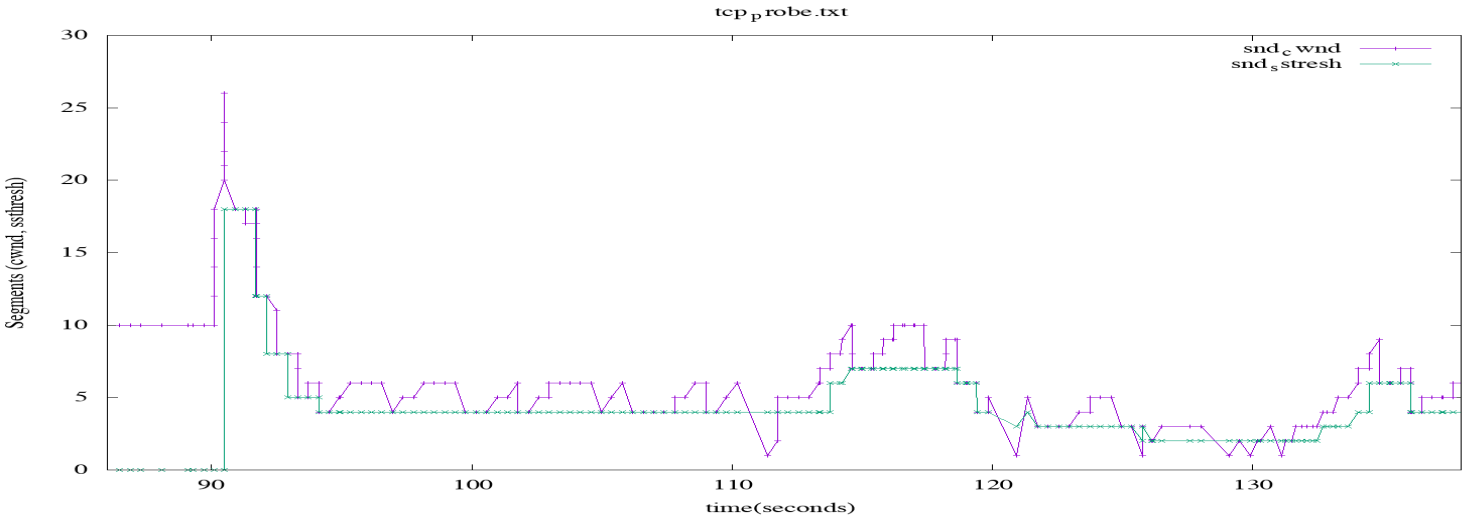

Figure 13: Cubic CWND/SSHTRESH sc. 2

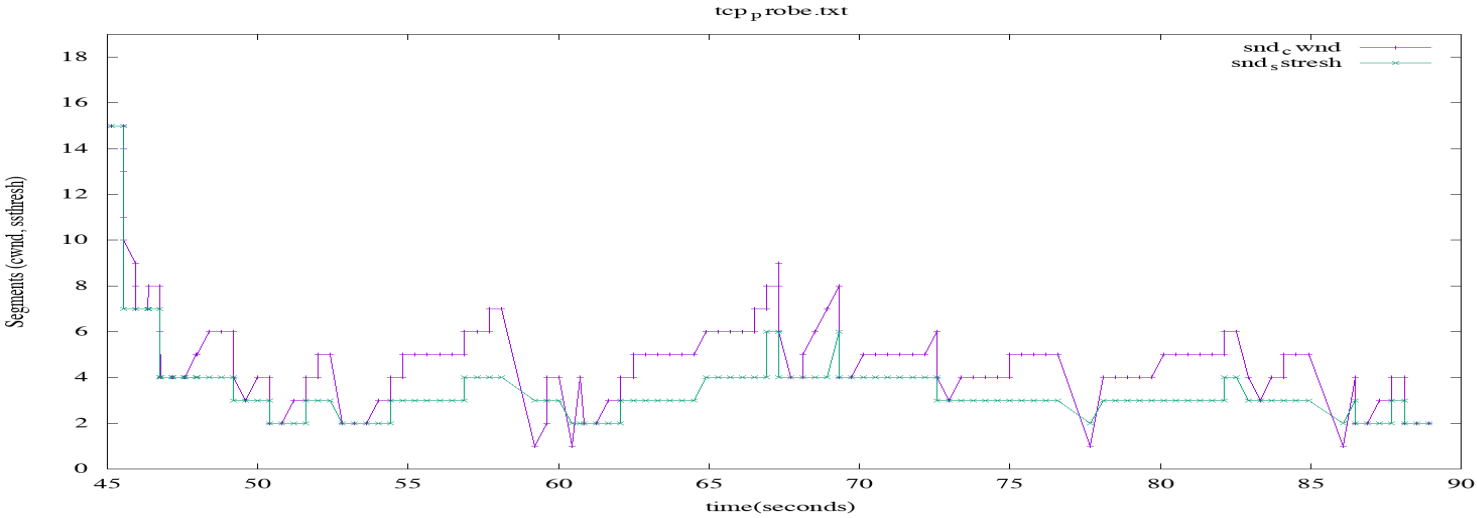

Figure 14: Vegas CWND/SSHTRESH sc. 2

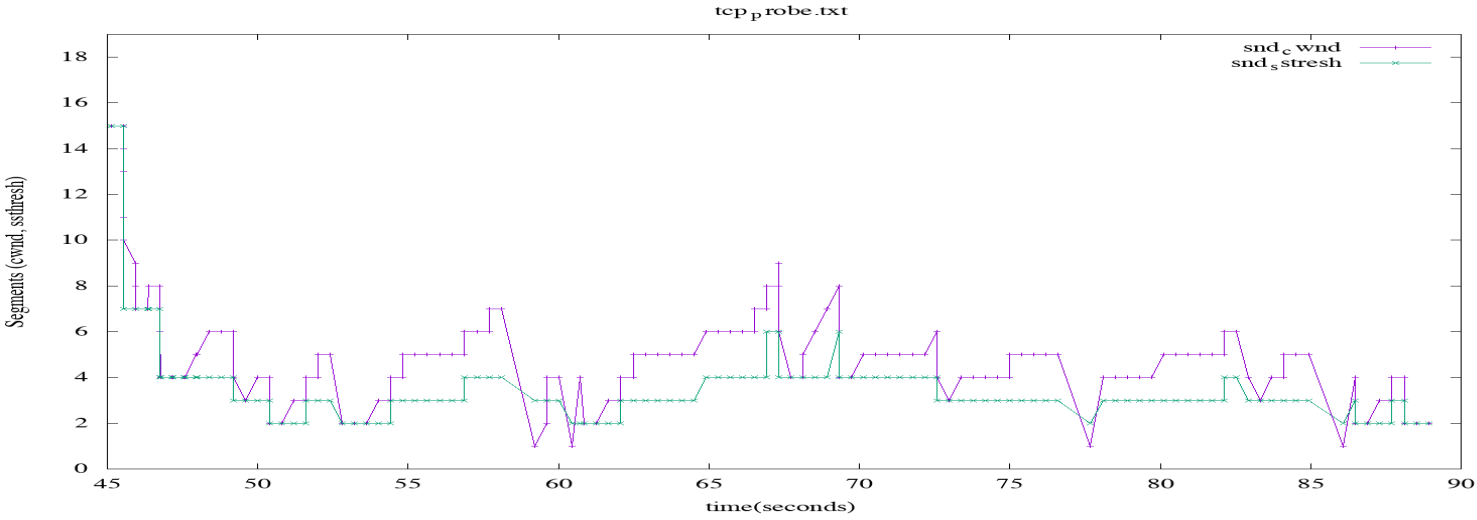

Figure 15: Cubic THR sc. 2

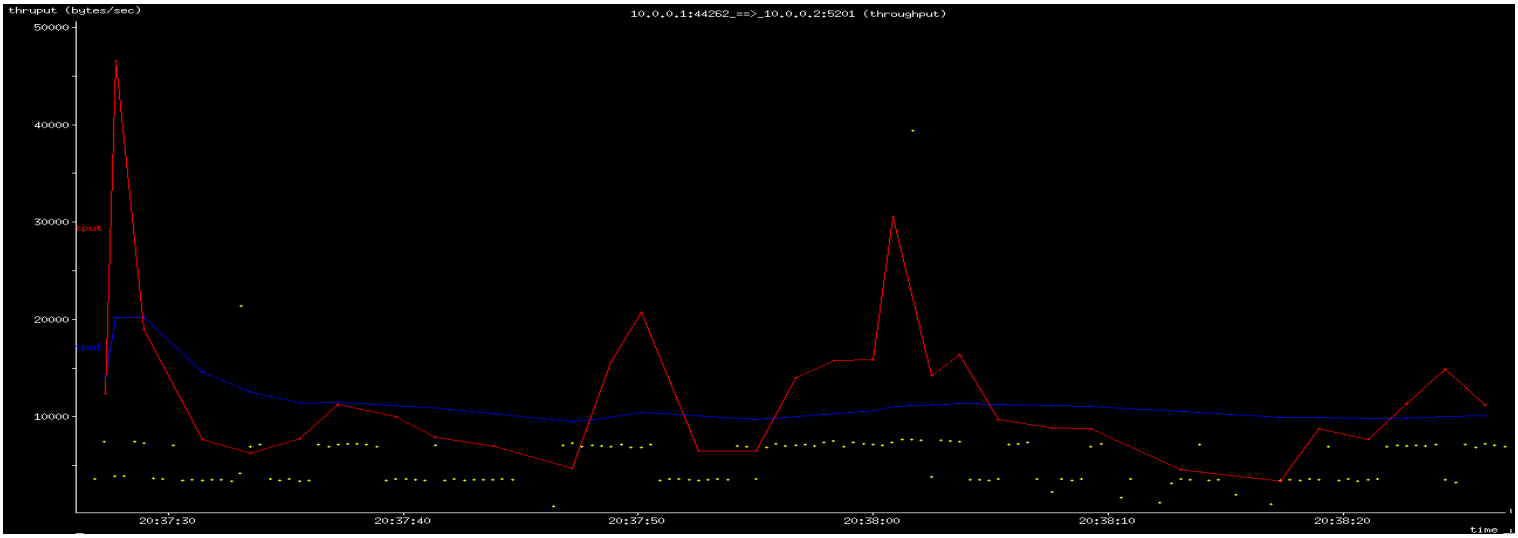

Figure 16: Reno THR sc. 2

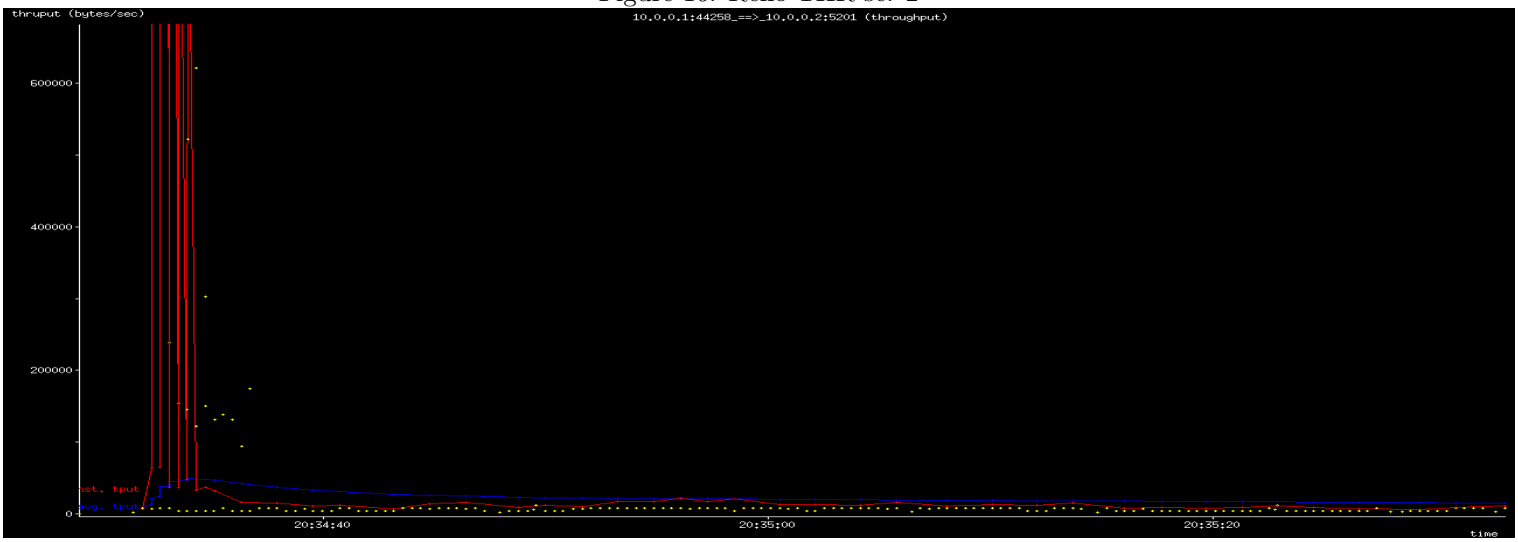

Figure 17: Vegas THR sc. 2

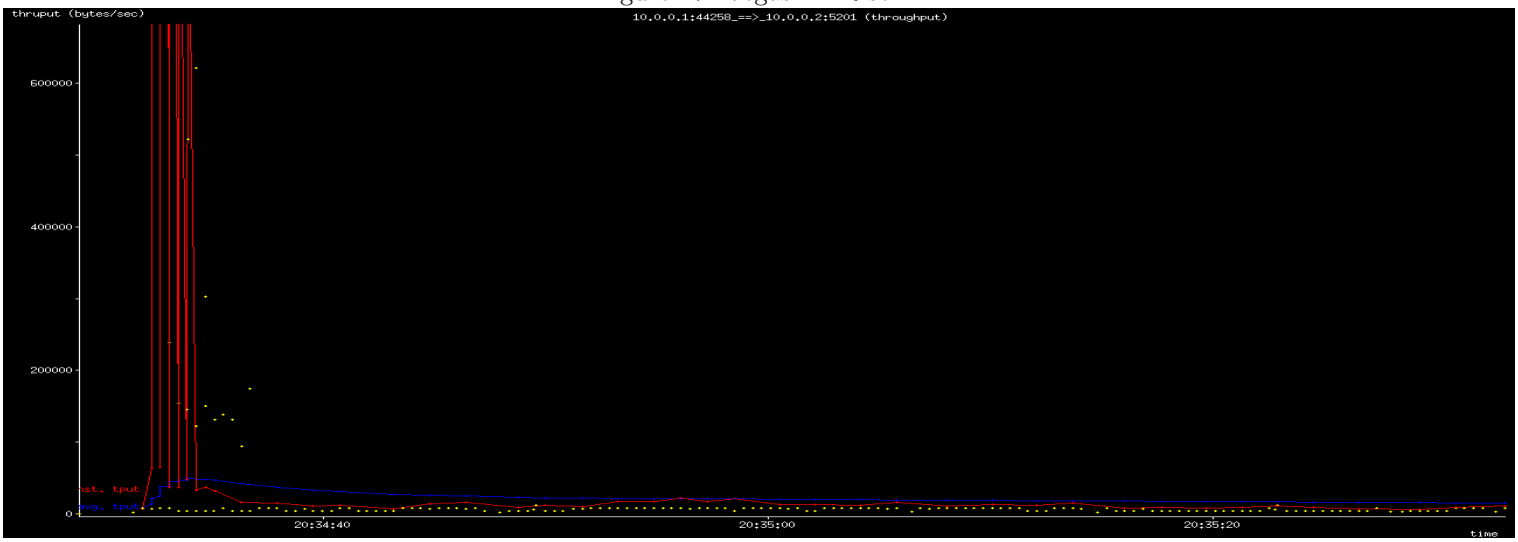

Figure 18: Reno RTT sc. 2

Scenario 4

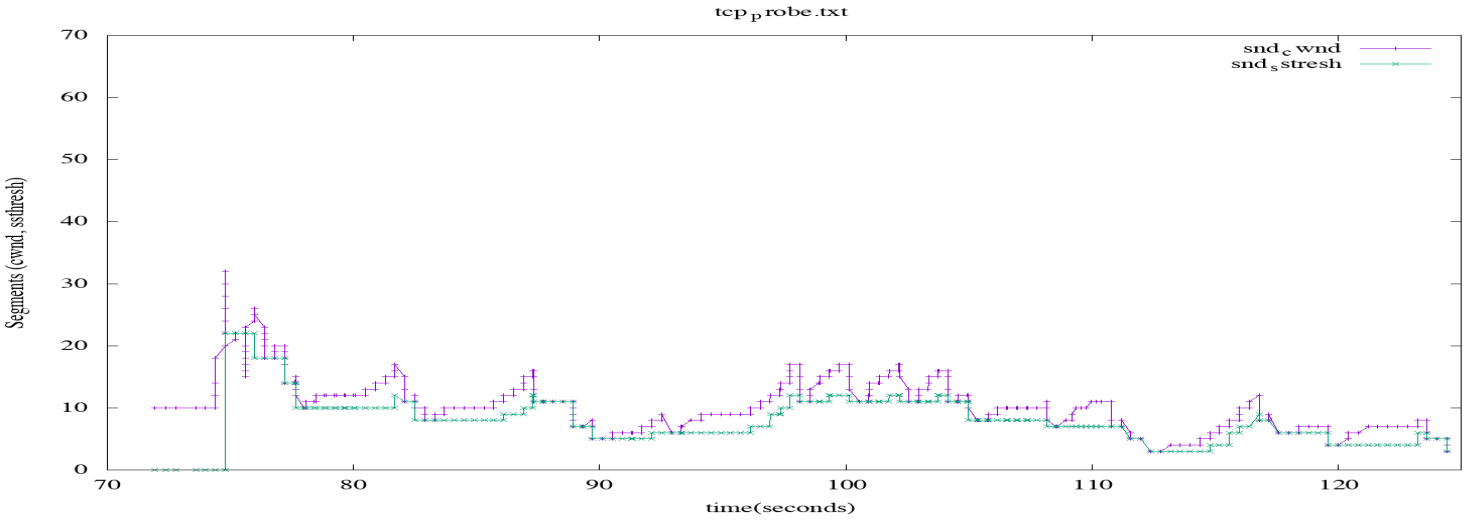

Figure 19: Cubic CWND/SSHTRESH sc. 4

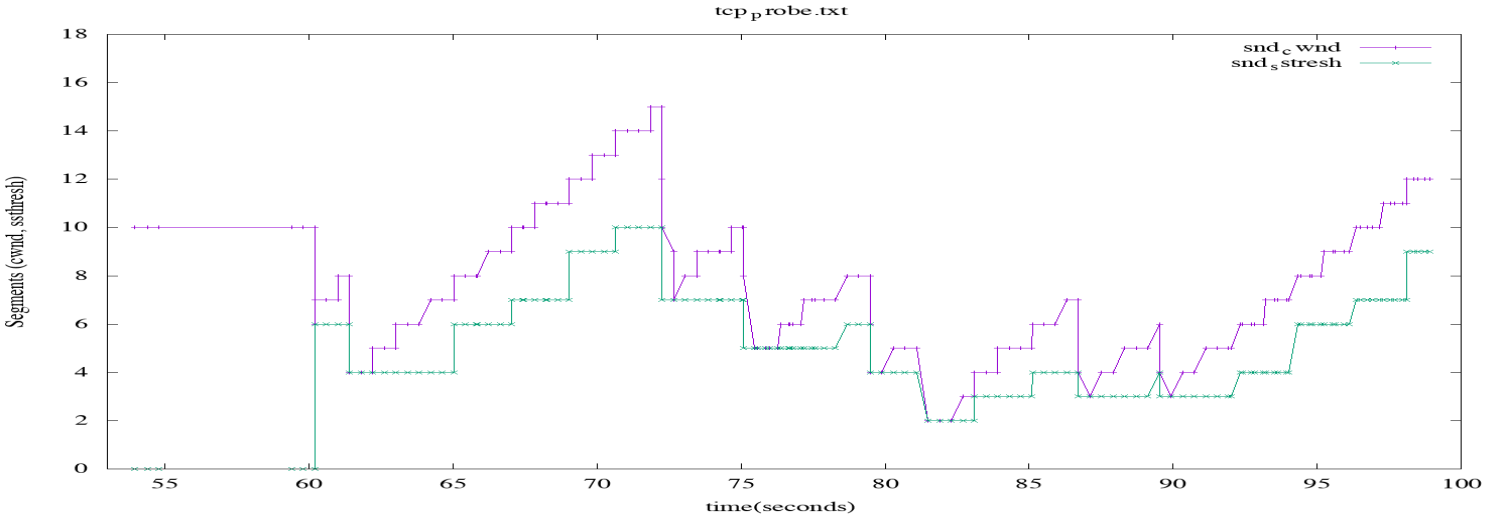

Figure 20: Reno CWND/SSHTRESH sc. 4:

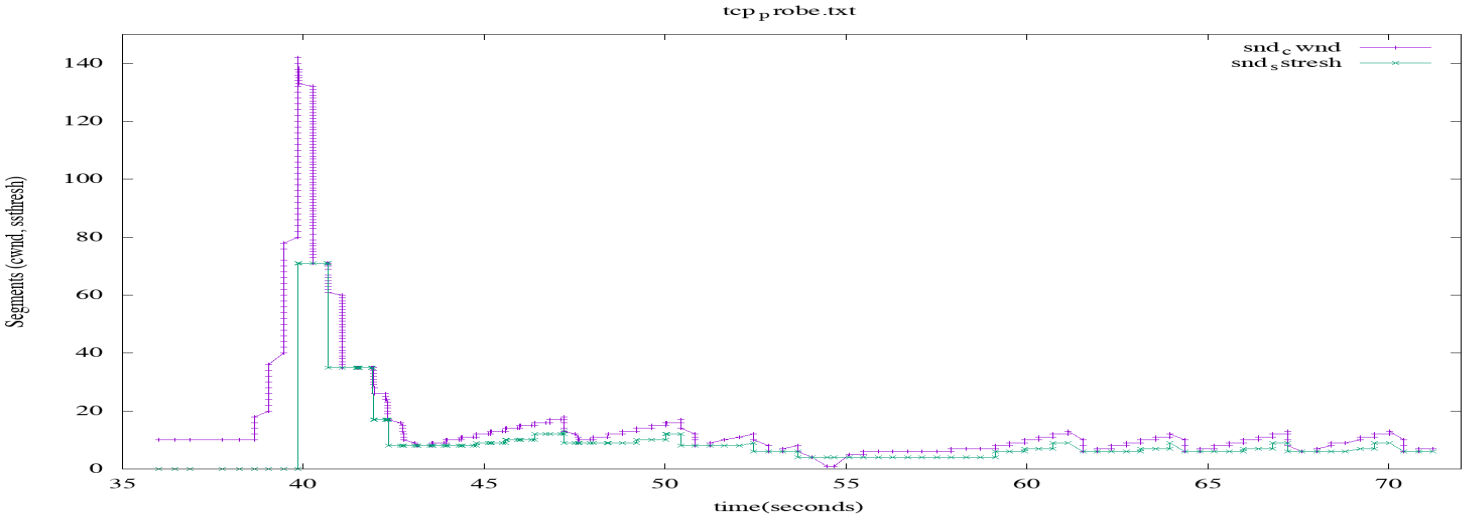

Figure 21: Vegas CWND/SSHTRESH sc. 4:

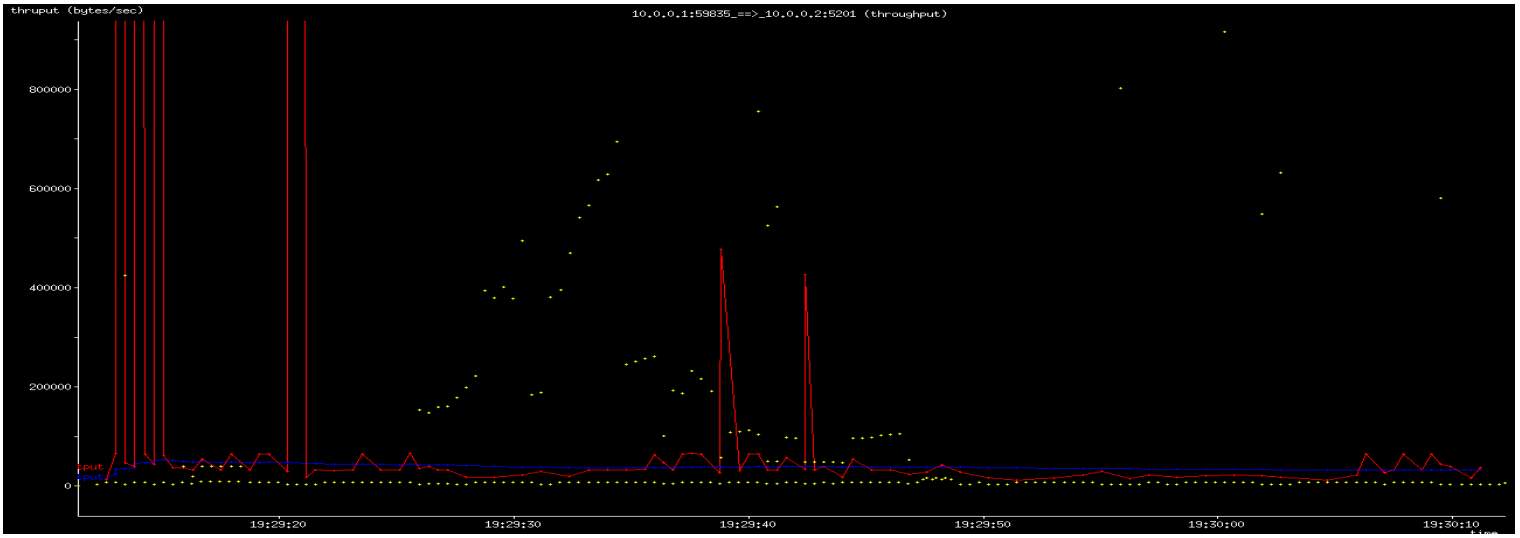

Figure 22: Cubic THR sc. 4

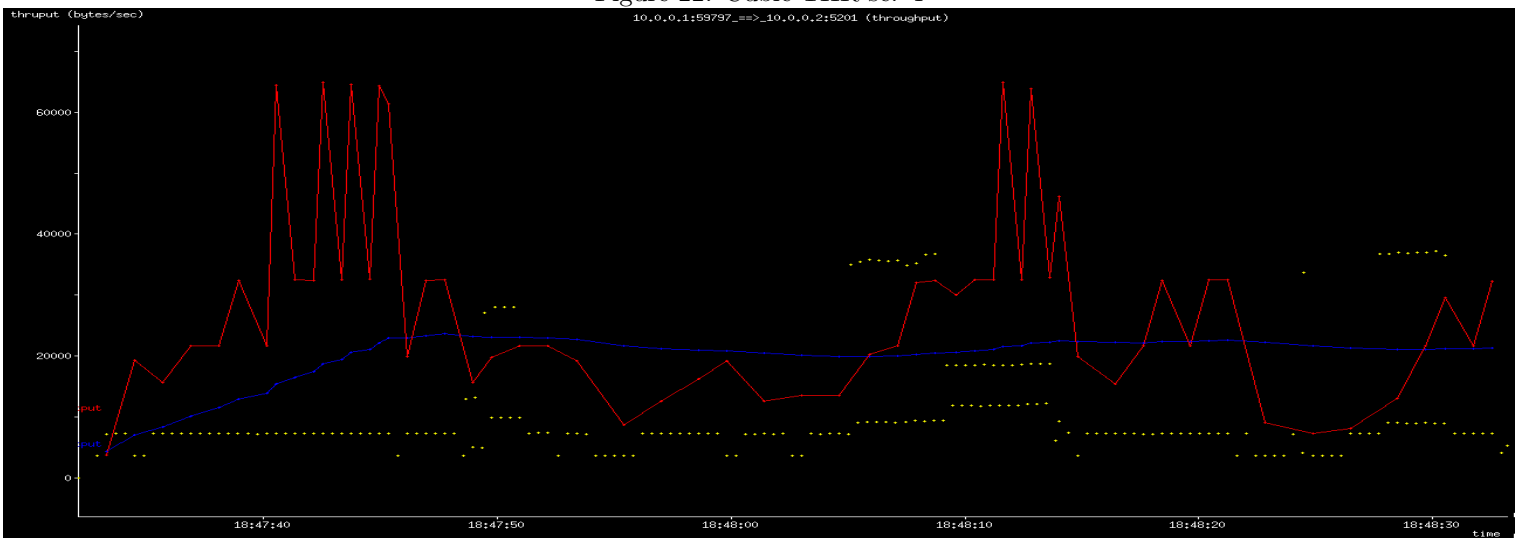

Figure 23: Reno THR sc. 4:

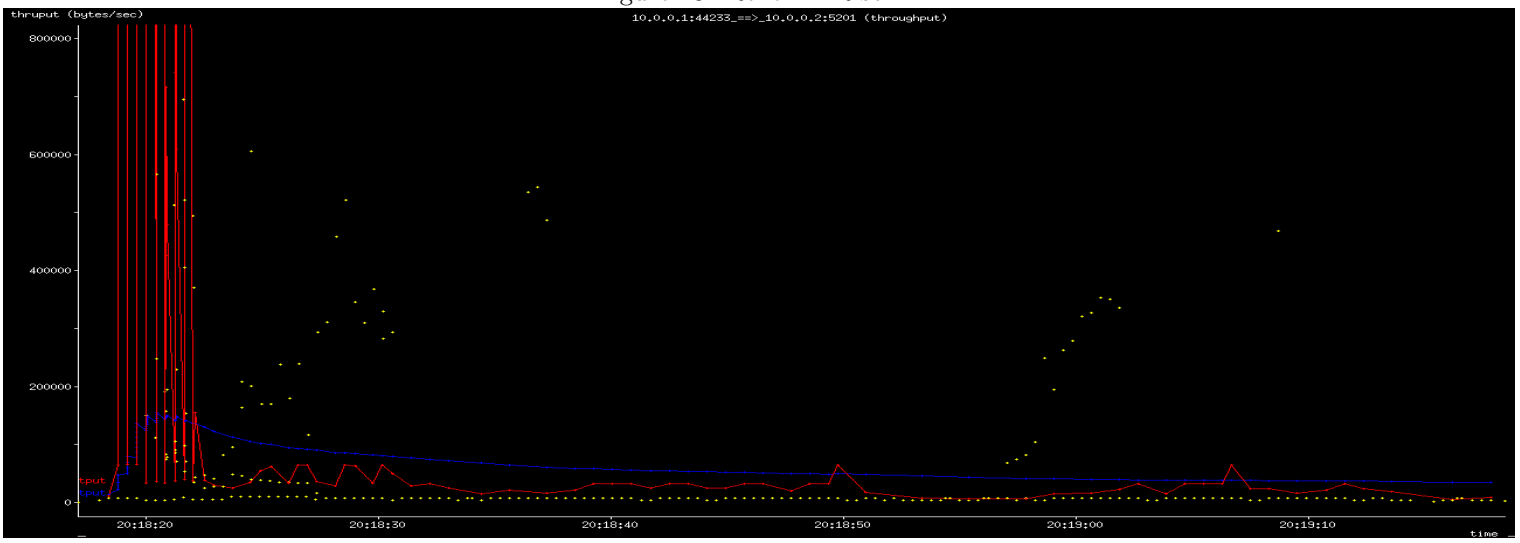

Figure 24: Vegas THR sc. 4:

Scenario 5

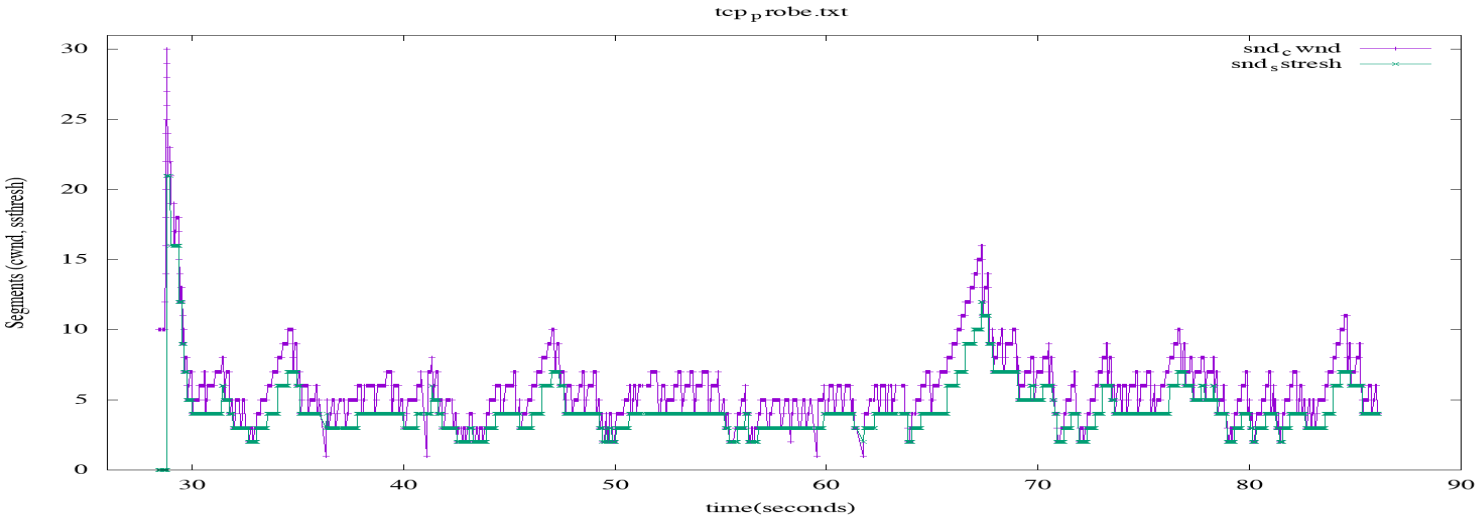

Figure 25: Cubic CWND/SSHTRESH sc. 5

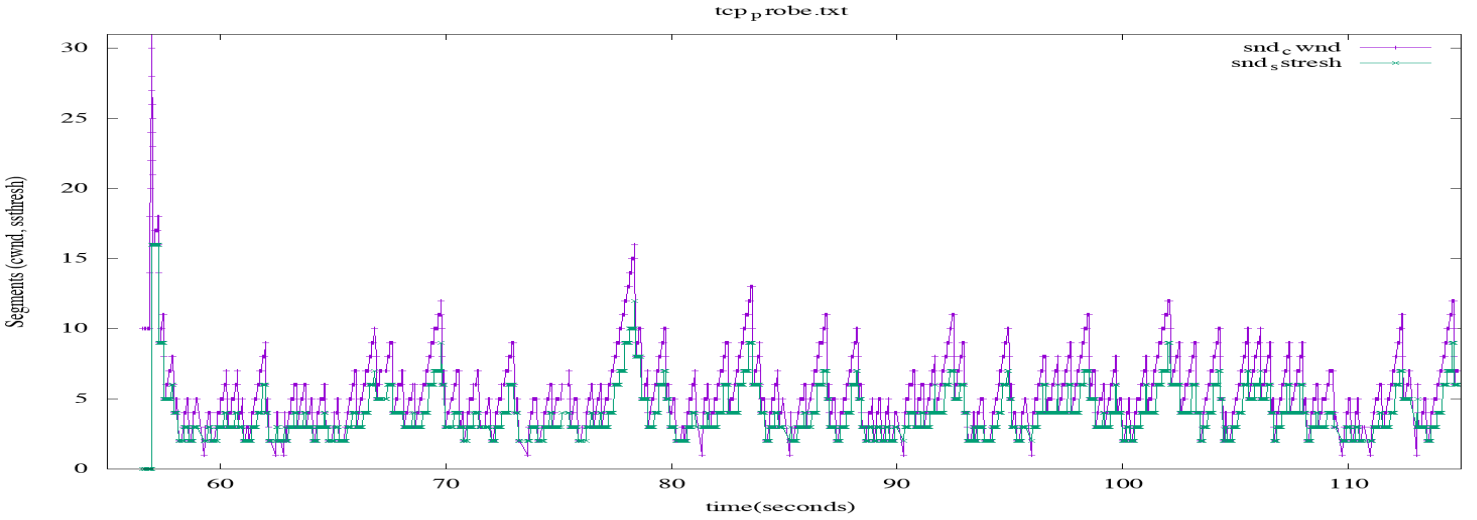

Figure 26: Reno CWND/SSHTRESH sc. 5:

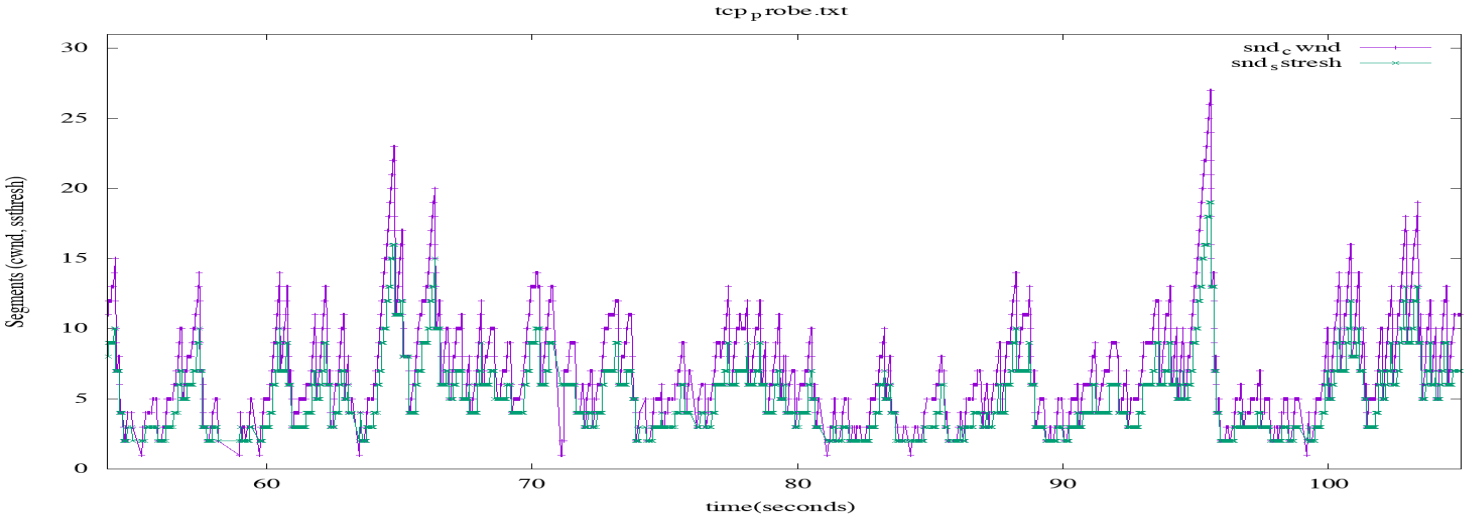

Figure 27: Vegas CWND/SSHTRESH sc. 5:

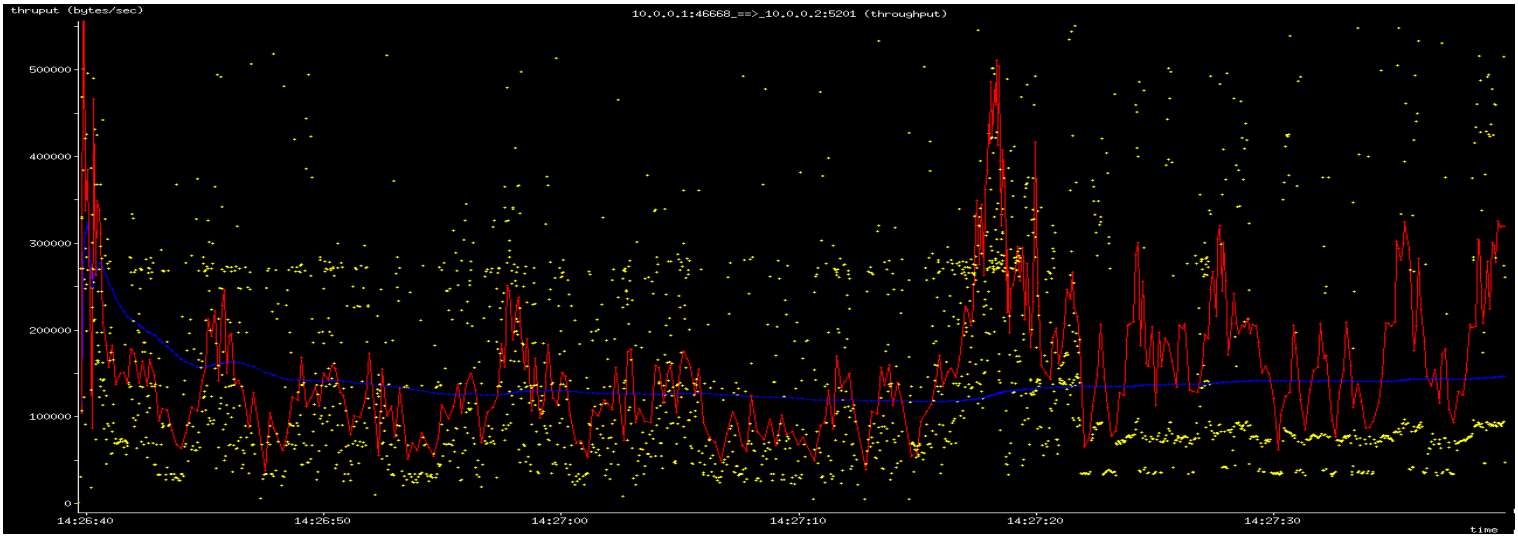

Figure 28: Cubic THR sc. 5

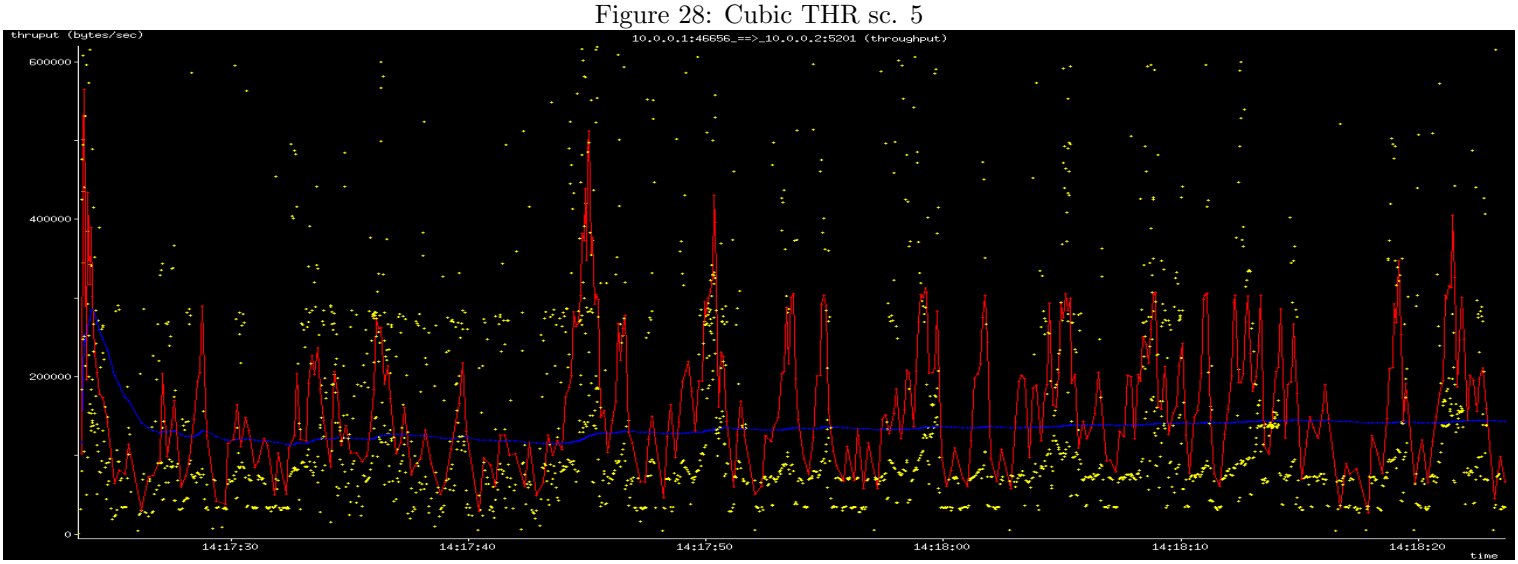

Figure 29: Reno THR sc. 5:

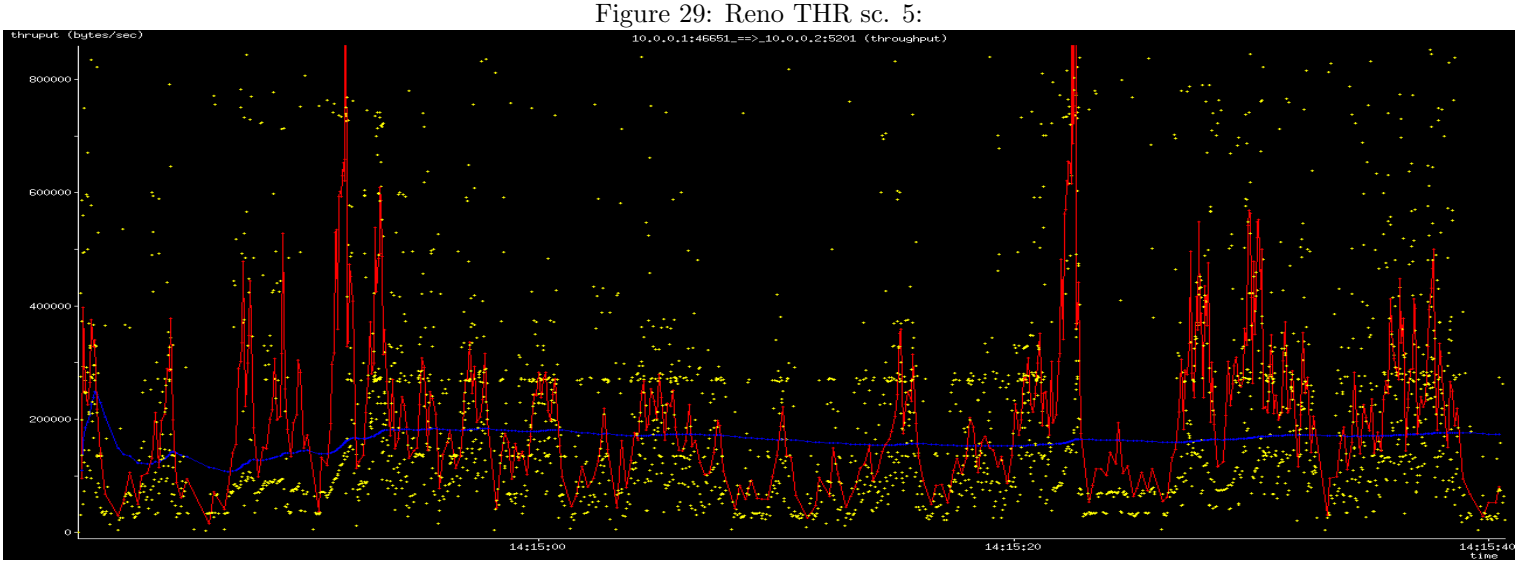

Figure 30: Vegas THR sc. 5: