Experimentation with Linux TCP Versions using Simulation

Javier Benitez Fernandez, Sowmya Ravidas
Aalto University School of Science
javier.benitezfernandez@aalto.fi, sowmya.ravidas@aalto.fi

Abstract

This report explains our experimentation with various Linux TCP Congestion control algorithms which was performed using Mininet emulation platform. The goal is to understand how different TCP algorithms work with varying network performance properties such as packet loss, bandwidth and latency. We also detail our analysis and findings from the measurement data which forms the basis for comparison between different TCP protocols.

KEYWORDS: TCP, Veno, Reno, Vegas, Mininet

1 Introduction

There are various TCP Congestion Control Algorithms and in this report we focus on TCP Reno, TCP Veno, TCP Vegas and TCP Cubic. We experiment how these TCP congestion control algorithms work with different network performance properties such as packet loss, bandwidth, latency etc. We use Mininet emulation platform to perform our experiment. In section 2, we will see our experimental setup and the conditions used for taking the measurements. In section 3 we will detail our analysis and Section 4 concludes the report.

2 Experimentation Setup

We use Mininet [1], a network emulation platform, with its default network to perform our experiment. The default network consists of one controller, one switch and two hosts as shown in Figure 1.

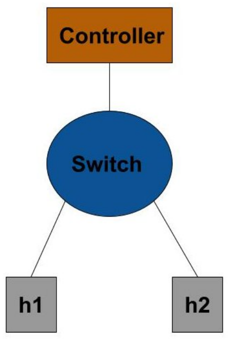

Figure 1: Default Network

The set up is done in the following steps.

1. Create a mininet Network

```
sudo mn --link tc,bw=b,delay=d,loss=l
```

This command creates a simple network consisting of one controller, one switch and two hosts. We added to this network different properties such as bandwidth, delay and packet loss. In this experiment, we used three different properties:

   (a) 100 Mbps of bandwidth, 10 ms of delay and 1% of packet loss.
   (b) 100 Mbps of bandwidth and 10 ms of delay, without packet loss.
   (c) 10 Mbps of bandwidth and 1 ms of delay.

2. Select TCP protocol

```
sudo sysctl -w net.ipv4.
tcp_congestion_control=protocol
```

This command changes the default TCP protocol. In this experiment, we used four different protocols: Reno, Veno, Vegas and CUBIC.

3. Prepare the environment. We prepared the environment creating a server in the first host h1 using the following command-

```
h1 iperf -s &
```

We enable a bulk transfer from the client(h2) to the server(h1) using iperf as follows.

```
h2 iperf -c h1
```

This step is repeated three or more times before taking the sample to ensure that samples are not contaminated.

4. Record data: To record TCP connections, we used tcp probe [2] which also records the state of TCP connection, capturing the congestion window and slow start threshold. Firstly, we need remove possible last recorded sessions

```
sudo rmmod tcp_probe
```

Secondly, we need to start a new session with a maximum of recorded data (1 MB in this experiment) and write the data in a file.

```
sudo modprobe tcp_probe full= 1
sudo cat /proc/net/tcpprobe > /path
```

5. Take the sample and remove the unnecessary fields. Our final data file consists of three fields: time, CWND and SSTHRESH and we further plot the graph of CWND/SSTHRESH with respect to time using MATLAB.

3 Analysis

3.1 Performance of Reno, Vegas, Veno and Cubic: Bandwidth=100Mbps, Delay=10ms and no packet loss

TCP Reno uses the Additive Increase Multiplicative Decrease algorithm (AIMD) for congestion control where the congestion window(CWND) is increased by a fixed amount for every acknowledgement and decrements only if it has received three or more ACKs [6] . In this scenario, the sample has no packet loss, and we see that the congestion window rise exponentially till it reaches the slow start threshold(SSTHRESH) value and the window size doesn't change after that (Figure 2).

On the other hand, TCP Vegas uses an algorithm that uses the difference in time between sending a packet and receiving the ACK to adjust the TCP window. Therefore, we can see a quick growth in the graph until ssthresh (1042) and later congestion increases which leads to delay and hence there is a decrease in the CWND as seen in Figure 2.

TCP Veno is an improved version of TCP Reno where the modifications are done in the congestion avoidance state. It also uses some properties of TCP Vegas in estimating the congestion in advance [4]. In this scenario, there is no packet loss and hence we can see the graph to be the same as TCP Reno (Figure 2).

In TCP Cubic, we can see an initial SSTHRESH of 1992 when CWND is 10. However, immediately after that we can see that CWND has a quick increment and SSTHRESH is adjusted accordingly. Cubic uses a cubic function time since the last congestion event and this permits it to adjust SSTHRESH to CWND. In the graph, we can see a rise without decrement, because TCP Cubic only reduces the window size during packet loss (Figure 2).

3.2 Performance of Reno, Vegas, Veno and Cubic: Bandwidth=100Mbps, Delay=10ms and with 1% packet loss

TCP Reno uses the AIMD for congestion control where the congestion window is increased by a fixed amount for every acknowledgement. If a packet loss is detected, Reno does Fast Retransmit where it retransmits the packet without waiting for the time out. During packet loss, the slow start threshold is set to half the value of the current CWND as can be seen from Figure 3. The new CWND is set to the the value that adds up SSTHRESH value and three times the packet size. It also increases the CWND by one for every duplicate ACK and when a non duplicate ACK arrives, the CWND value is set to SSTHRESH. This process is known as the Fast Recovery [6]. The problem with TCP Reno is that it detects the congestion at a later stage.

In TCP Vegas, the algorithm detects the possibilities for congestion at an early stage and hence reduces the sending rate. If the difference between the expected throughput and the actual throughput is less than the threshold, then TCP vegas increases the congestion window linearly till the next Round Trip Time. And if the difference is more than the threshold, then the congestion window is decreased linearly.

TCP Veno also uses the AIMD same as TCP Reno. The difference is during the congestion avoidance state, the CWND is increased by 1 for every 2 Round trip times unlike Reno where we increase for each RTT. This can be easily seen when we compare the graphs of Reno and Veno in Figure 3. One of the drawbacks of Reno is the inability to identify if the packet loss is due to congestion or due to network fault. Veno modifies the multiplicative decrease algorithm to solve this problem. If it is due to network loss, the new SSTHRESH is set to 4/5 times the CWND and if it is due to congestion, the SSTHRESH is reduced half the CWND size. This helps in distinguishing congestive and non-congestive state [8]. In this scenario, the packet loss is due to congestion and we can see that the window size of Veno is reduced to half as in Reno.

TCP Cubic also adjusts its window size depending the threshold just like TCP Vegas. There is a binomial increase in CWND when there is no congestion and we can see a multiplicative decrease in the CWND during a packet loss (Figure 3).

3.3 Performance of Reno, Vegas, Veno and Cubic: Bandwidth=10Mbps, Delay=1ms and no packet loss

In TCP Reno, we can see an initial SSTHRESH of 4032, very high because we do not save metrics, with a CWND of 10 (Figure 4). We can see a slow exponential increment of CWND before second 6. Later, with a period of packet loss we can see a momentary decrement of SSTHRESH to 1003 while CWND decrease slowly to 1006.

We are not sure if this packet loss is due to congestion or due to network fault. However when compared to the graph of TCP Veno, we can see a strong similarity in the results. Hence, we can conclude that, it is the packet loss due to the congestion as the CWND value of TCP Veno is reduced to half as seen from Figure 4. In this scenario, TCP Veno is better because it takes less time and hence small delay when compared to Reno.

In TCP Vegas, we can see an initial SSTHRESH of 44032 and a CWND of 10 as seen from Figure 4. CWND quickly rise to 28 before second 3, after a small delay CWND and SSTHRESH are reduced to 15. It is adapting to the congestion condition, and hence the delay increases. TCP Vegas adapts to this condition with decrement in CWND and SSTHRESH values. This property cannot be seen in CUBIC, Reno or Veno.

TCP CUBIC increments progressively for both SSTHRESH and CWND. Unlike Vegas, Cubic just reduces the CWND with packet loss. In this case, we have no packet loss but we have a huge delay marked by congestion. CUBIC adapts to this condition with a slow growth as seen from Figure 4 and also the graph is similar to the graph of

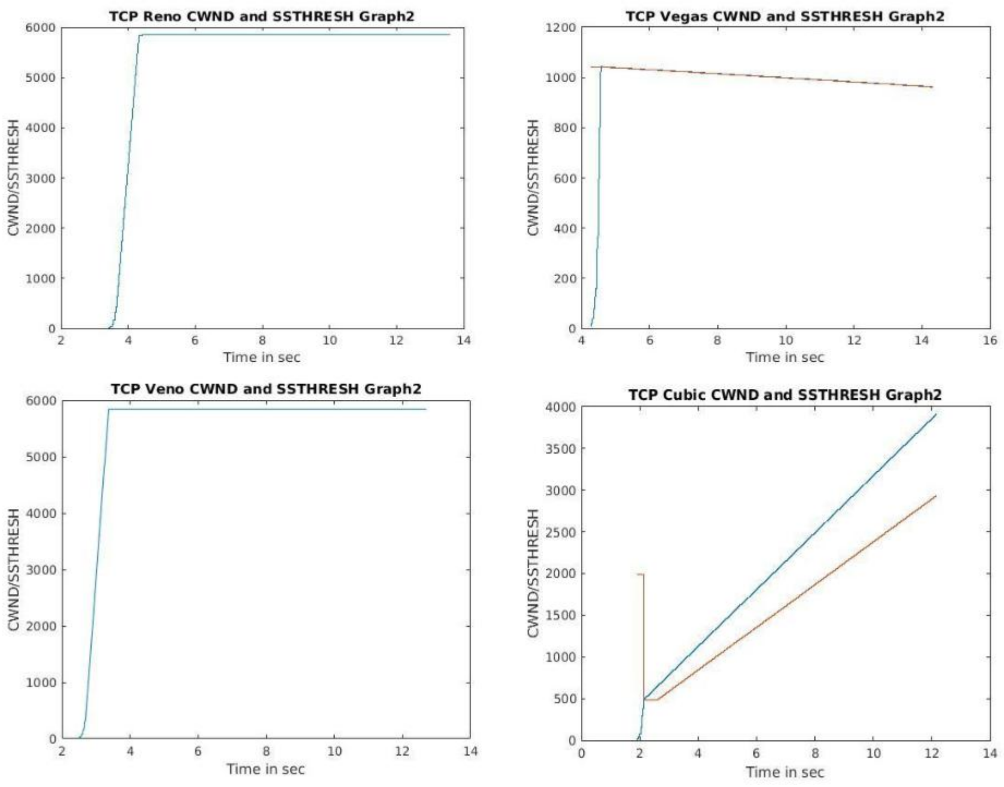

Figure 2: Analysis without packet loss

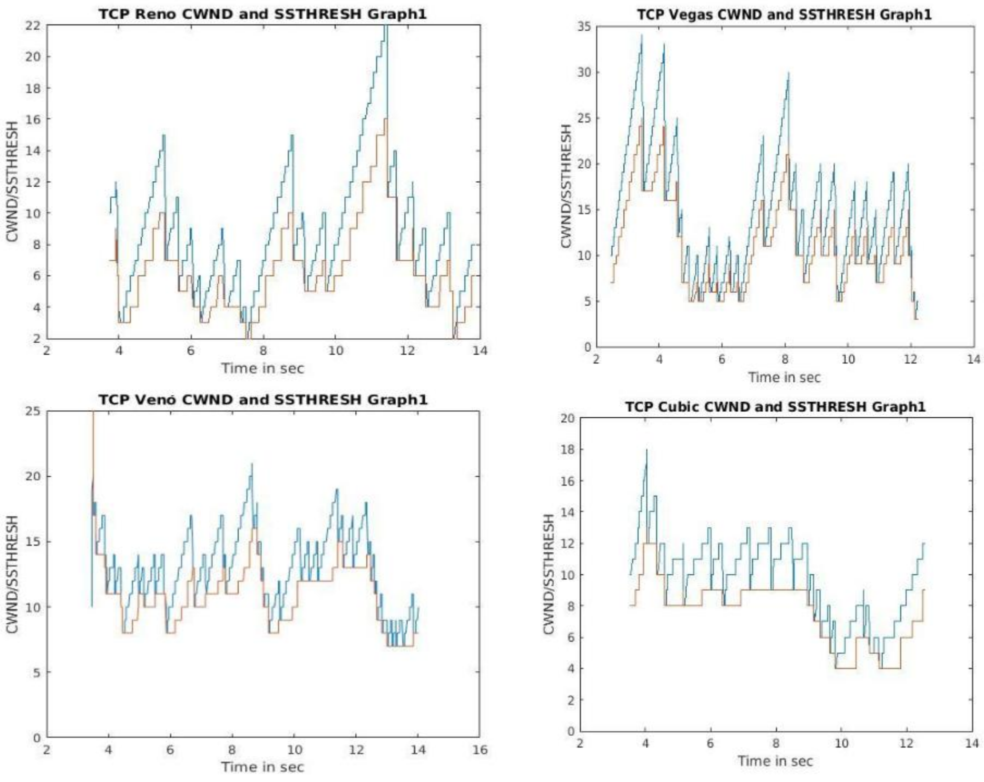

Figure 3: Analysis with packet loss

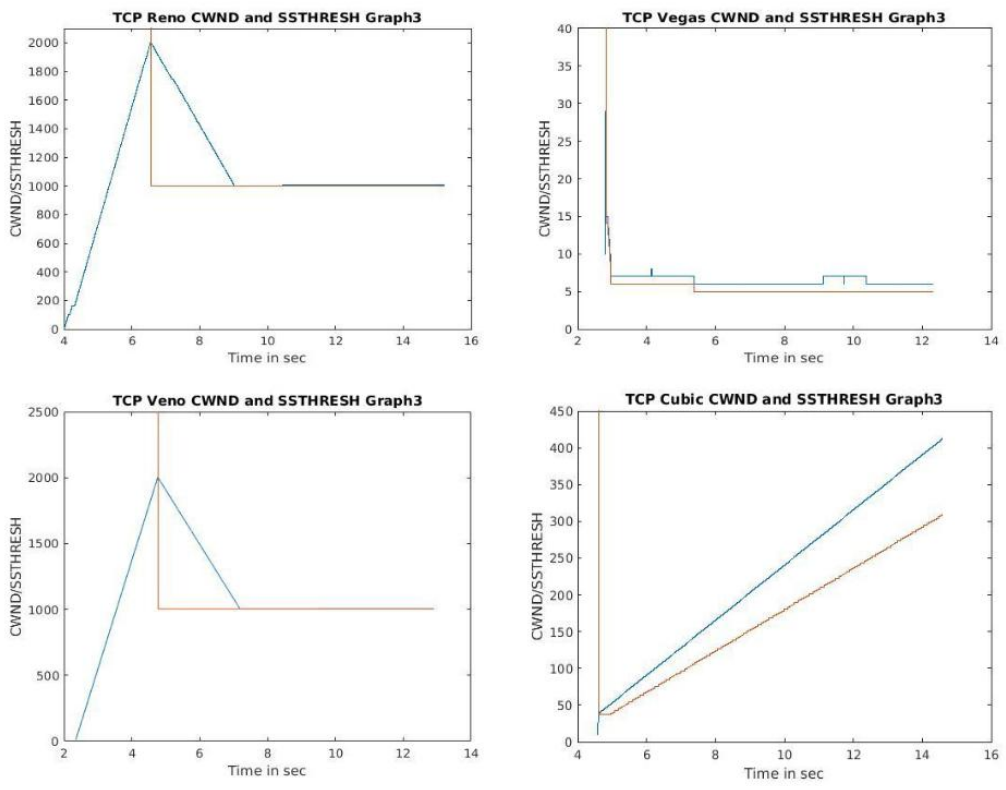

Figure 4: Analysis with reduced delay and bandwidth

Cubic in Figure 2.

4 Conclusion

In this report, we performed experiments to analyze the performance of different Linux TCP congestion control algorithms- TCP Reno, TCP Vegas, TCP Veno and TCP Cubic. We observed each of these with properties such as packet loss, bandwidth and delay. We observed that TCP Veno shows almost similar behavior as TCP Reno in cases where packet loss is due to congestion. TCP Vegas might be useful to detect congestion at early stage, however the performance is less since it reduces the window size gradually even when there is potential to send more data. TCP Reno and Veno are older and have a abrupt adaptation, Vegas was the one that obtained the best performance for limited bandwidth and for small networks. However, CUBIC was better with high bandwidth and larger networks where CUBIC adapts faster than Vegas. To conclude, the environments and samples show us that neither TCP protocols can be decided as the best option for all the considered scenarios.

References

[1] Mininet. WWW Introduction to Mininet: `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet`.

[2] Tcp probe. WWW TCP Testing: `http://www.linuxfoundation.org/collaborate/workgroups/networking/tcp_testing`.

[3] Tcp protocol part 1. WWW Slides of TCP lecture, Computer Networks- Avanced Features, Aalto University: `https://mycourses.aalto.fi/pluginfile.php/101877/mod_folder/content/0/Transport-part1.pdf?forcedownload=1`.

[4] C. P. Fu and S. C. Liew. Tcp veno: Tcp enhancement for transmission over wireless access networks. *Selected Areas in Communications, IEEE Journal on*, 21(2):216–228, 2003.

[5] S. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[6] L. A. Marrone, A. Barbieri, and M. Robles. Tcp performance. *Journal of Computer Science & Technology*, 13, 2013.

[7] J. Mo, R. J. La, V. Anantharam, and J. Walrand. Analysis and comparison of tcp reno and vegas. In *IEEE INFOCOM*, volume 3, pages 1556–1563. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 1999.

[8] C. Zhang, C. P. Fu, M.-T. Yap, C. H. Foh, K. Wong, C. T. Lau, and M. Lai. Dynamics comparison of tcp veno and reno. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 3, pages 1329–1333. IEEE, 2004.