Evaluating the Impact of Time Delays and Start Sequence for Effective Congestion Control Using TCP Reno, Westwood and Vegas

Sumin Yi and Rahul Gomes
Department of Mathematics and Computer Science
Minot State University
Minot, ND 58701
sumin.yi@ndus.edu and rahul.gomes@ndus.edu

Abstract

Transmission Control Protocol (TCP) provides stable data delivery among applications by allocating bandwidth [1]. When congestion occurs on a network, transmission delays and packet loss can happen. Therefore, TCP ensures a stable network by controlling the congestion [8]. Since each TCP variant uses different methods to measure the congestion, it is important to figure out how it cooperate and without affecting the performance of the other TCP variants.

This paper compares three TCP variants that use congestion control algorithms. It emphasizes on the time intervals when running multiple TCP protocol transmissions. Experiments are conducted by varying start times of each source node as well as the order of precedence for the protocols. Simulations were conducted on the NS2 Platform installed on Ubuntu 14.04.

Experiments reveal the greedy nature of Reno [3] when it starts well ahead of other TCP algorithms. However, when Vegas and Westwood start well ahead of Reno in our experiments, the network appears optimized, and the impact of Reno is reduced.

1 Introduction

The Transmission Control Protocol (TCP) is a connection-oriented protocol in the Transport layer [3]. The transport layer is in charge of establishing communication between applications and delivering data. There are many services that are provided by TCP in the transport layer. These include ensuring the end-to-end connection between the source and the destination, and handling error corrections in order to provide a solid connection [8]. It can provide reliable transport layer to many protocols on the Internet as integrity is enforced in TCP. TCP uses congestion control algorithms to reduce delay and data loss due to network congestion. It puts flow control and congestion control into use to manage both sender and receiver which in turn controls the data that flows through the network.

Congestion control uses many methods to solve the bottleneck on the network. TCP Reno utilized packet loss, its main congestion detection method [9] while TCP Vegas uses increased Round Trip Time (RTT) to detect the congestion before packet loss actually occurs [5]. On the other hand, TCP Westwood uses both packet loss and RTT estimation to estimate the packet rate as it calculates the congestion window [2] [3]. There are many ways to measure and respond to congestion. However, the most important thing is to find the optimum way to utilize each attribute of TCP variant when using multiple variants together.

TCP Reno, TCP Vegas, and TCP Westwood are used to test the importance of having time intervals between TCP protocol transmissions. The essential part of the experiments is to set various time intervals between the source nodes in addition to having different arrangements of the protocols. The experiments were tested on NS2 simulations and GNUPLOT was used to visually describe the results. The start time of each experiment was adjusted by 0.4 sec, 1 sec and 2 secs. The duplicated process was done by changing the arrangement of the protocol to validate the prominence of greediness.

The rest of the paper is organized as follows. Section 2 explains the three TCP variants that were used in these experiments. Previous researches conducted on the comparison of TCP congestion control algorithms are in section 3 for a better understanding of this paper. Details of the experiments including the method used are given in section 4. The results and demonstrations are listed in Section 5. Section 6 has a summary of the results.

2 Background

TCP Reno, Vegas, and Westwood are subjects of this research because of their correlations. TCP Reno and Westwood share similar congestion control algorithms: as Westwood is a modified version of Reno, being sender-side only [3]. They both use fast recovery to reset the slow start threshold (ssthresh) and congestion window (cwnd) after loss happens [10]. TCP Vegas has its own congestion control algorithms distinct from Reno but it shares similarities with TCP Westwood by to use RTT measuring method to estimate the bandwidth. Having this set of three makes it easier to compare the different congestion control algorithms used in the experiments.

2.1 TCP Reno

TCP Reno resembles TCP Tahoe which being that both of them are the earliest model of congestion avoidance algorithms used. TCP Reno uses slow start until it hits the threshold and runs additive increase methods same as TCP Tahoe [3]. Additive increase means increasing congestion window by 1 when there are no losses in the past transmission. However, the difference between TCP Reno and Tahoe occurs when they receive triple duplicate acknowledgments (ACK). TCP Tahoe will cut its cwnd size to 1 and enter slow start [3]. TCP Reno will perform fast recovery meaning it will set the ssthresh into the half the size of cwnd and wait for an ACK of transmit window before going into congestion avoidance [9]. If TCP Reno faces time-out instead of triple duplicate ACKs, slow start will begin from 1 just like TCP Tahoe. TCP Reno employs passive congestion indication which can make the congestion worse on the network. Packet loss detects the network congestion of TCP Reno.

2.2 TCP Vegas

TCP Vegas uses precise RTT calculation to detect the available bandwidth. It calculates the difference between expected flows and actual flows. It measures the minimum RTT to estimate the RTT of a network without congestion, if the network is not congested, the difference should be close to zero [9]. Packet delay determines the rate of sending packets instead of packet loss which means it detects the congestion before packet loss because of increased RTT. In cases that packet loss does occur, the packet sending rate gets lowered linearly as well. TCP Vegas gets compared to TCP Reno many times as they represent contrasting congestion control method [5] [9]. Vegas encounters fairness problems when it collides with Reno due to its main congestion avoidance mechanism. TCP Vegas tries to maintain the small queue size whereas TCP Reno would fill up the queue since its packet loss happens due to overflowing buffer. Therefore, Reno is deemed aggressive as it does not give space for other connections to take the bandwidth. Once TCP Reno takes up the buffer, Vegas will perceive it as congestion and will drop the cwnd of its own.

2.3 TCP Westwood

TCP Westwood uses end-to-end bandwidth estimation which updates ssthresh and cwin when packet loss emerges [6]. It scans the ACK stream to have a better slow start. Agile probing helps to converge the bandwidth faster to a more appropriate ssthresh. Persistent Non-Congestion Detection identifies unused bandwidth and invokes agile probing. TCP Westwood is the refinement of TCP Reno and keeps the loss based congestion avoidance mechanism by dropping cwnd size when queue capacity is less than transit capacity [3]. Unlike TCP Reno that just decreases its cwnd to half, TCP Westwood is able to choose the ssthresh and cwnd based on the congestion it is experiencing at the time. TCP Westwood also uses the RTT estimation method from TCP Vegas as it is very efficient when handling high bandwidth connections. While TCP Vegas estimates its bandwidth by cwnd/RTT, TCP Westwood goes further and uses bandwidth estimation to reduce cwnd based on the network present network traffic [3]. Although it implanted TCP Reno's mechanism, it is not as aggressive as Reno as it also has a feature to estimate bandwidth [7].

3 Previous Work

Work on TCP congestion control was done earlier in [4]. The authors evaluate 13 TCP variants and examine how TCP variants are compatible with each other. Each TCP variant uses miscellaneous congestion control algorithms and it makes a huge change to how network traffic conveys. The authors of this research measured how congestion control algorithms fight for network resources. Congestion window changes as the congestion condition changes in the network – it gets smaller when the network is having heavy traffic and gets bigger when the traffic is less to prevent a TCP connection to overwhelm the network. They also found out that some of the TCP variants can cooperate with other TCP variants whereas some took too much bandwidth and overwhelmed the other TCP variants. They mention TCP Vegas as the algorithm which gives up bandwidth the most easily hence the most sensitive to network congestion. On the other end, some algorithms did not share the bandwidth even when competing against similar types and some did share the bandwidth only among the similar types. One of the factors that some of the congestion control algorithms did not share the bandwidth is because they were meant for high-speed networks with large RTTs.

Similar TCP variants were used in [7] to evaluate and compare three control algorithms: Westwood+, New Reno and Vegas. NS2 simulations and live internet measurements were the methods for the simulations and scenarios were set to examine goodput, fairness, and friendliness of each algorithm. The research demonstrates how Westwood+ TCP is friendly toward New Reno and adjust better in bandwidth allocation while Vegas is impartial but has problems grabbing its bandwidth share when operated with Reno or reverse traffic as its RTT-based congestion detection mechanism. When dealing with wireless links that are affected by losses not dues to congestion, Westwood+ did the best on utilizing the network. Westwood+ provides well-improved utilization of loss links. Apart from that, Westwood+ also showed better goodput compared to New Reno when the pipe size is larger than a few segments in real life measurements.

4 Experimental Setup

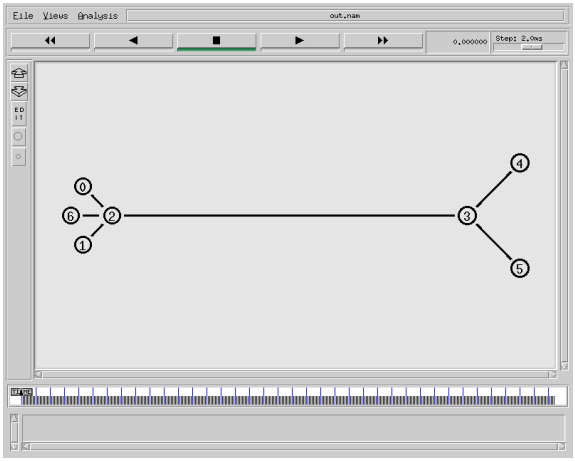

Figure 1: nam set up in NS2 simulator

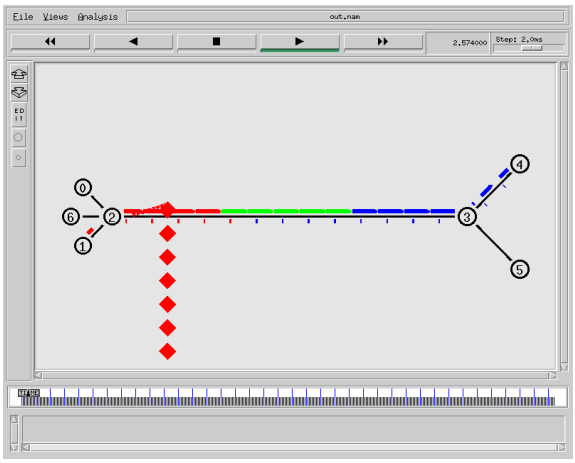

Figure 2: Visualizing congestion and packet drop (red)

Experiments were conducted in the Linux system Ubuntu 14.04 using the ns2 simulator. The nam set up file is visualized in figure 1. Two sender nodes and two receiver nodes were connected with one connection and new sender node was added after existing sender nodes started sending the packets. Original two sender nodes were set to have the same type of TCP congestion control algorithm and the extra node was altered among three TCP variants to spot the differences. Each TCP variant was tested against other two TCP variants and also itself. The first source node starts at 0 seconds and the next node with the same TCP variant starts after a set time interval. The last node starts after the same time interval which could be possibly another type of TCP variant. The time intervals were 0.4 seconds, 1 second and 2 seconds.

The same experiments were repeated with a set of TCP variants with the change in time intervals. The result of the 3 test runs is summarized in the tables in the next section. The time intervals were varied to test how it could influence the sharing of bandwidth. The order of protocols was changed to test analyze how precedence of protocols affect the greediness. The procedures and conditions of the experiments were identical except for the set of TCP congestion control algorithms and time interval.

5 Test and Results

This section discusses the output of the tests conducted with different time intervals and order of precedence of the TCP algorithms. We show the influence of these factors on Reno, Vegas, and Westwood. Graphs are shown for time intervals 0.4 s and 2 s only to stress the differences. However, the tables show the results for all the time intervals.

5.1 Reno

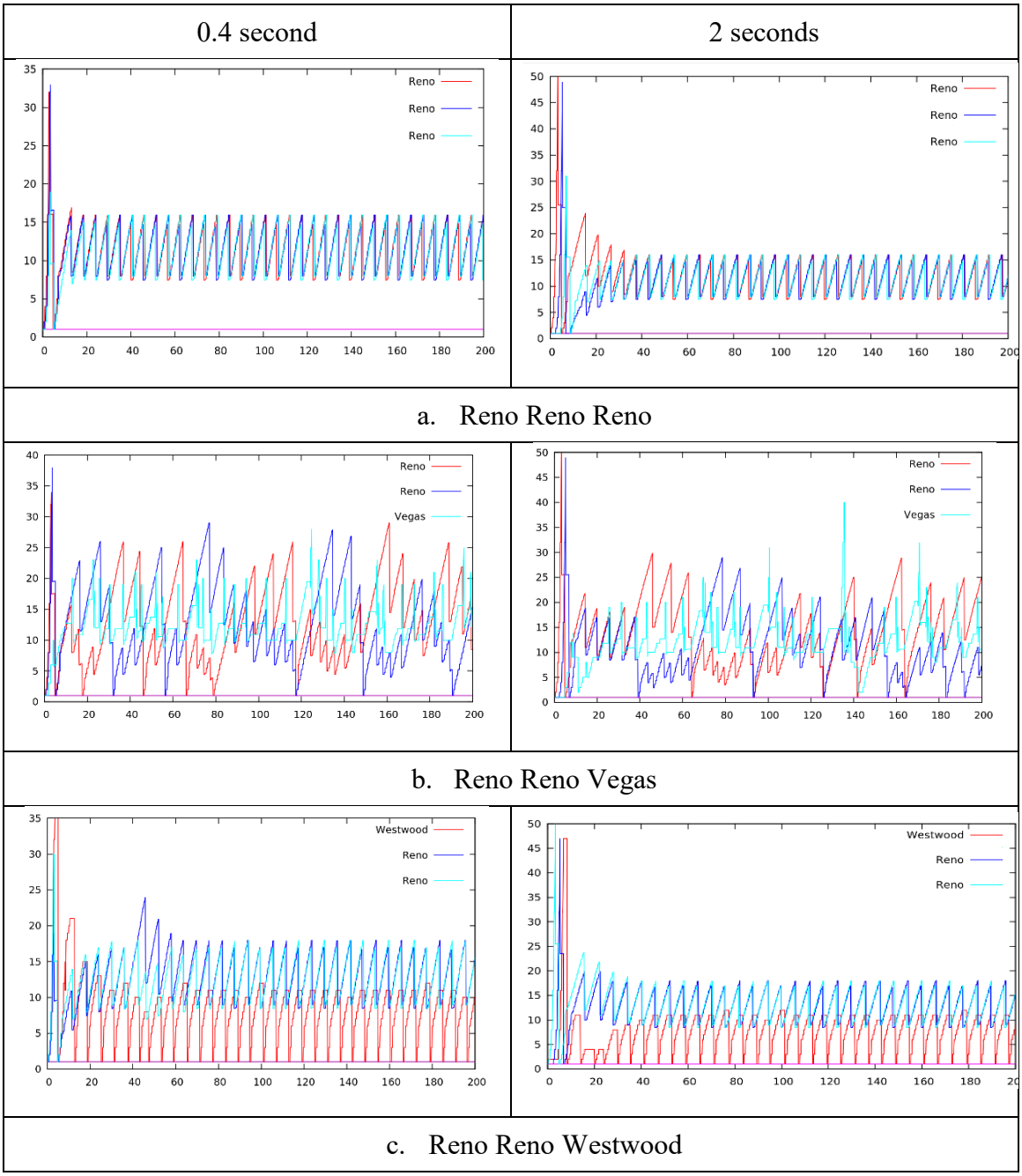

Figure 3: Reno based experiments

Interval / TCP	0.4s	1.0s	2.0s
Reno	11.51	10.99	11.99
Reno	11.43	11.62	11.00
Reno	11.21	11.59	11.13
a. Reno Reno and Reno			
Reno	13.10	12.74	13.96
Reno	12.68	12.66	11.32
Vegas	12.25	12.20	11.99
b. Reno Reno and Vegas			
Reno	12.44	12.46	13.22
Reno	12.73	13.23	12.86
Westwood	8.47	8.22	7.84
c. Reno Reno and Westwood			

Table 1: Average window size on Reno based experiments

Figure 3 illustrates the graphs when TCP Reno is used to making two of the sender nodes connections. Each graph shows that when there is a distinct time interval, it takes a little time for the later nodes to adjust to the traffic and balance the throughput. However, once the later nodes get adapted to the system, they show the fixed pattern of graphs in Figure 3.a and Figure 3.c. It is displayed in Figure 3.a that Reno is compatible with itself. Both of the graphs from time interval 0.4 second and 2 seconds represent the steady graphs from all three nodes. Table 1.a shows equal distribution of window sizes among each node. It explains that the congestion control algorithm of TCP Reno does not apply when the other connection is using the same algorithm. Figure 3.b shows that TCP Reno and Vegas are crisscrossed throughout the transit. As appeared in the graphs, the time interval did not make much difference. TCP Reno does not take up the majority of the bandwidth and coexisted with Vegas, the average in Table 1.b proves that contrary to the expectation of Reno overtaking the bandwidth due to its greedy nature, they coexisted. Figure3.c. demonstrates that TCP Reno and Westwood have stable graphs. Time interval did not make any distinctions except for the very beginning of the graphs, which was caused by the adaptation of the last node. According to Table 1.c, TCP Reno took up approximately 1.5 times or more bandwidth compared to Westwood. It is because TCP Westwood estimates the bandwidth and lower the cwnd while Reno was overflowing the buffer.

5.2 Vegas

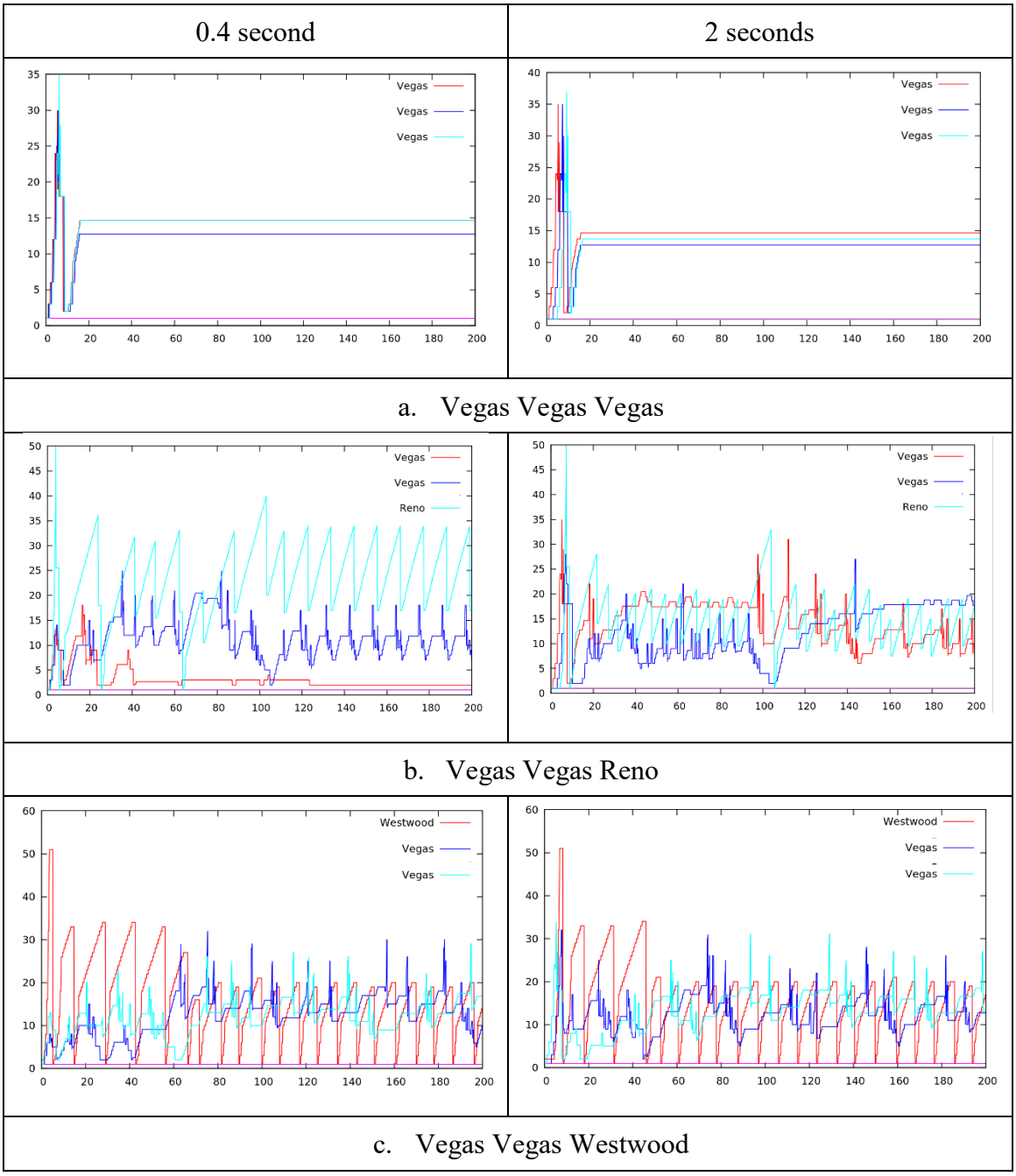

Figure 4: Vegas based experiments

Interval / TCP	0.4s	1.0s	2.0s
Vegas	14.24	10.71	14.30
Vegas	12.46	18.70	12.43
Vegas	14.22	11.67	13.22
a. Vegas Vegas and Vegas			
Vegas	3. 26	12.71	13.94
Vegas	11.23	14.18	11.96
Reno	23.09	13.20	14.66
b. Vegas Vegas and Reno			
Vegas	11.84	13.82	13.15
Vegas	12.31	12.68	12.04
Westwood	16.18	13.32	15.34
c. Vegas Vegas and Westwood			

Table 2: Average window size on Vegas based experiments

Figure 4 presents the graphs of TCP Vegas being the dominant congestion control algorithm in the experiments. All three comparison in Figure 4 well describes the characteristics of each TCP congestion control algorithm. Figure 4.a shows that TCP Vegas can coexist with the connections of its own algorithms. The time interval, in this case, did not affect how the bandwidth is distributed. Table 2.a also displays the fair distribution of average cwnd except for the 1 second time interval. Yet, as the other two intervals do not carry the same result, it is assumed that fluctuation of the TCP Vegas caused the difference. Figure 4.b exhibits symbolic differences between the time interval. With smaller time interval, TCP Vega's graphs are noticeably lower than TCP Reno's graph. However, with the bigger time gap, three nodes seem to balance the window size. Especially, TCP Reno is within the range of two TCP Vegas's graphs. TCP Reno clearly shows greedy nature at first, but as the time intervals got greater the window size balances out. Table 2.b shows the significant number of differences from 0.4-second interval and 1-second interval. Figure 4.c showcases no influence of the time interval. Two graphs have a similar pattern and window sizes. Table 2.c also shows that overall, TCP Westwood takes a little more window size compared to TCP Vegas despite the late start. It is not assumed to be meaningful in this research as it does not show the sequence of the effects of the time interval.

5.3 Westwood

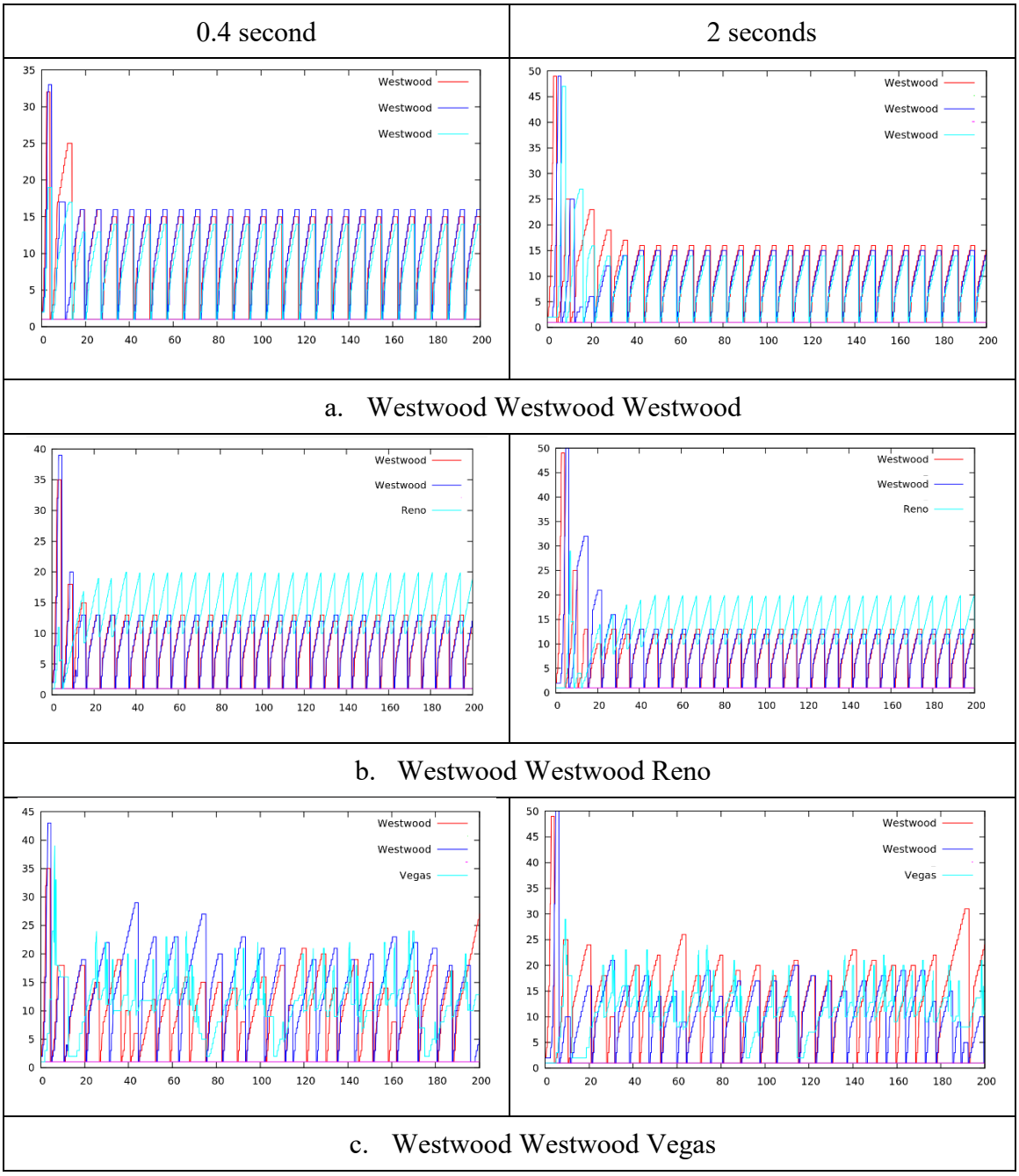

Figure 5: Westwood based experiments

Interval / TCP	0.4s	1.0s	2.0s
Westwood	11.05	9.68	11.99
Westwood	11.43	10.29	10.43
Westwood	9.33	12.53	9.76
Westwood Westwood and Westwood			
Westwood	9.29	9.98	9.28
Westwood	9.22	9.29	10.11
Reno	14.03	13.61	13.56
Westwood Westwood and Reno			
Westwood	11.33	11.98	14.91
Westwood	14.54	14.01	11.30
Vegas	11.39	10.94	11.12
Westwood Westwood and Vegas			

Table 3: Average window size on Westwood based experiments

Figure 5 portrays the TCP Westwood based experiments and how the graphs are formed with Westwood being the original TCP congestion control algorithm. Figure 5 shows the least influence of time intervals among the three experiments based figures. Other than the beginning of each graph, the patterns between the small time interval and large time interval simulate. Figure 5.a illustrates the graphs of three TCP Westwood connections. Comparing two different graphs does not provide a piece of extra information on how time interval has affected the graphs. They depict the typical graph of TCP Westwood and shows that it can cooperate with itself. Table 3.a also shows roughly the same allocation on window size. Figure 5.b does show that TCP Reno takes more bandwidth than TCP Westwood albeit it started late. However, Table 3.b elaborates on precise average window size and it shows consistent reduce on TCP Reno's window size. Although the differences are small, it is yet meaningful result in a way that it shows the same pattern of decrease. Figure 5.c demonstrates well-balanced graphs in bandwidth take wise which means TCP Westwood and TCP Vegas can coexist. Two graphs look like they do not have order and spontaneous. However, Table 3.c shows equal distribution of the window size and no pattern.

6 Conclusion

In this paper, we attempted to see the impact of time intervals and order of precedence on TCP congestion control. We found that there are three main results made after the tests. First of all, having a noticeable time interval helps to stabilize the window size. As TCP Reno has greedy behavior, it tends to take a lot more window size when executed with multiple congestion control mechanisms. However, starting TCP Vegas and Westwood well ahead from Reno improves the intake of window size. Secondly, the order of TCP variants precedence has an impact on consumption of window size. Even though same time intervals were used, running TCP Reno before Vegas or Westwood did not balance the window size. Lastly, taking time interval and order of precedence into measure can be a good solution to establish a fair TCP connection. Providing a time delay for Reno could be beneficial to tackle this greedy nature and ensures TCP fairness across multiple protocols. Further research in this field would be examining the optimal time interval between the different TCP transmissions.

References

[1] Allman, M., Paxson, V., & Blanton, E. (2009). *TCP congestion control* (No. RFC 5681).

[2] Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M. Y., & Wang, R. (2002). TCP Westwood: end-to-end congestion control for wired/wireless networks. *Wireless Networks*, *8*(5), 467-479.

[3] Dordal, P. (2017). An introduction to computer networks.

[4] Esterhuizen, A., & Krzesinski, A. E. (2012). TCP Congestion Control Comparison. *SATNAC, September*.

[5] Feng, W. C., & Vanichpun, S. (2003, January). Enabling compatibility between TCP Reno and TCP Vegas. In *2003 Symposium on Applications and the Internet, 2003. Proceedings.* (pp. 301-308). IEEE.

[6] Gerla, M., Sanadidi, M. Y., Wang, R., Zanella, A., Casetti, C., & Mascolo, S. (2001). TCP Westwood: Congestion window control using bandwidth estimation. In *GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No. 01CH37270)* (Vol. 3, pp. 1698-1702). IEEE.

[7] Grieco, L. A., & Mascolo, S. (2004). Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control. *ACM SIGCOMM Computer Communication Review*, *34*(2), 25-38.

[8] Kurose, J. F., & Ross, K. W. (2011). *Computer networking: a top-down approach* (pp. 607967-5). Addison Wesley.

[9] Mo, J., La, R. J., Anantharam, V., & Walrand, J. (1999, March). Analysis and comparison of TCP Reno and Vegas. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)* (Vol. 3, pp. 1556-1563). IEEE.

[10] Stevens, W. R. (1997). TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms.

[11] Wang, R., Valla, M., Sanadidi, M. Y., Ng, B. K., & Gerla, M. (2002). Efficiency/friendliness tradeoffs in TCP Westwood. In *Proceedings ISCC 2002 Seventh International Symposium on Computers and Communications* (pp. 304-311). IEEE.