Performance Analysis of TCP Variants

Luke Janik, Jaison Titus
Northeastern University, Boston MA

***Abstract*---Transmission Control Protocol (TCP) is the most widely used protocol in the Internet protocol suite. TCP ensures that packets between hosts on an IP network are delivered reliably and in order. Since its inception, several TCP variants have been introduced including TCP Tahoe, Reno, New Reno, and Vegas. Each of these variants handles congestion control and avoidance differently, and it is the goal of this paper to analyze the differences in performance among these algorithms. We accomplish this by simulating each variant while changing the load on the network and observe the behavior. We perform a qualitative analysis by calculating latency, drop rate, and throughput for each experiment.**

I. INTRODUCTION

While the original TCP design was reliable, it would not perform well in today's large and congested networks. New versions of the algorithm have been proposed that add techniques to ease congestion. We will analyze the behavior of four of these variants: TCP Tahoe, Reno, NewReno, and Vegas. TCP handles congestion using the Additive Increase / Multiplicative Decrease (AIMD) algorithm. With Tahoe, if four Acknowledgements (ACKs) for the same packet are received, a *fast retransmit* is performed which sets the slow start threshold to half the congestion window and resets to the slow start phase. Reno improves upon Tahoe by introducing *fast recovery*, which halves the congestion window instead of setting it to 1 like in Tahoe. This effectively allows it to skip the slow start phase. New Reno builds on Reno by always keeping the transmit window full. It does this by sending a new unsent packet from the end of the congestion window for each duplicate ACK that is received. TCP Vegas behaves differently from the first three variants because it detects congestion from increasing Round Trip Times (RTTs) of packets instead of dropped packets.

In this paper we will examine which TCP variant performs the best in the presence of congestion by measuring latency, drop rate, and throughput. We performed three experiments to analyze the TCP behavior. The first experiment involves one TCP stream in the presence of a UDP stream sending at a constant rate. We varied the timing and bit rate of the UDP stream to vary the tests. The second experiment compares the variants for fairness by introducing two TCP streams. The final experiment measures TCP performance when the queuing algorithm is changed.

We found that in congested networks, TCP Vegas had the least latency and the lowest drop rate. In Experiment 2 we found that TCP New Reno did not behave fairly with other TCP variants. Finally, in Experiment 3 we concluded that RED queueing allows for greater use of the available bandwidth.

II. METHODOLOGY

We used NS-2 to build the network to perform the simulations. NS-2 is a well-regarded discrete-event computer network simulator that allows the user to easily change the topology, parameters, and variants. Figure 1 shows the network topology that was used for all three experiments. Each link in the shown topology has a bandwidth of 10Mbps. Each experiment uses a variation of this network setup by adding different sources and sinks depending on what is trying to be achieved.

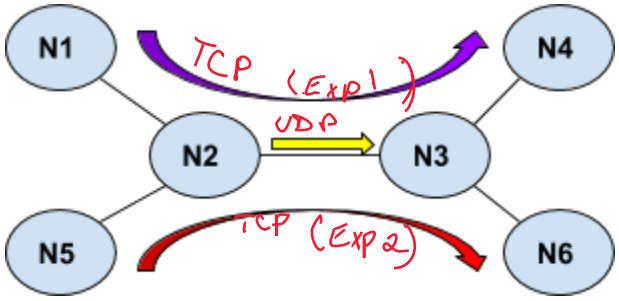

Figure 1. The network topology for the three experiments.

The general set up is a UDP connection from Node 2 to Node 3 that transmits at a constant bit rate. In Experiment 1, we add a TCP link from Node 1 to Node 4, and in Experiment 2 we add a second TCP link from Node 5 to Node 6. In Experiment 3, we have the same setup as Experiment 1 with 1 TCP link from Node 1 to Node 4, but vary the queuing algorithm that is implemented. Experiment 1 and 2 use DropTail as the queuing algorithm for all of its links.

We used Python3 to write scripts to easily tweak the parameters of the NS-2 program in order to generalize the behavior of TCP outside the simulated environment. For example, we used the Python script to vary the bit rate from 1-15 Mbps. This was done to see how the TCP variants behaved in different levels of congestion. Next, we changed the start time of the CBR flow. We started the flows concurrently, then delayed CBR to 2ms and 5ms, which allows the TCP to get out of its slow start phase before the CBR flow is introduced into the network. We used the script to change the TCP variant itself depending on the experiment. Lastly, we used Python3 to parse and aggregate the data from the NS-2 trace files into 'csv' files. We used the aggregated data to create charts in Google Sheets. The experiment purpose and specific design will be explained in its respective section.

III. EXPERIMENT ONE

Experiment 1 is designed to analyze the performance of the four TCP variants. The network is setup as shown in Figure 1 with a constant CBR with a source at N2 and sink at N3. A single TCP stream is set with a source at N1 and a sink at N4. We ran 45 tests per variant for a total of 180 tests for this experiment. For each TCP variant, we changed when the CBR flow started (0, 2, and 5 seconds after the TCP began). For each of those start times we changed the CBR (every integer bit rate from 1 Mbps to 15 Mbps for the CBR). Nodes N5 and N6 were not used for this experiment.

We tested the drop rate, latency, and throughput. The results show that the drop rate is better (lower) in TCP Vegas. Figure 2.1 shows that all of the variants performed well in the network until the CBR rate was increased to more than 7 Mbps which is when we can infer that congestion on the link started. When the CBR rate is increased to 13 Mbps, TCP Vegas drops only 11.5% of the packets, whereas the other variants drop 15.4% (Tahoe) to 15.9% (Reno) and 16.4% (New Reno). Vegas uses RTT to detect congestion in the network and lowers the congestion window accordingly before packets are dropped. All three other variants detect congestion through packet loss, so it makes sense that they have similar packet drop rates.

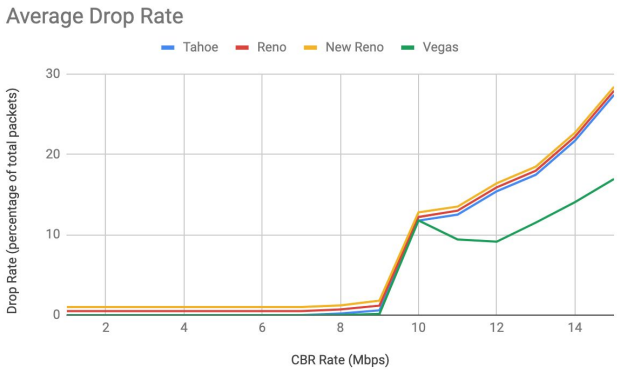

Figure 2.1 Average drop rate for Experiment 1.

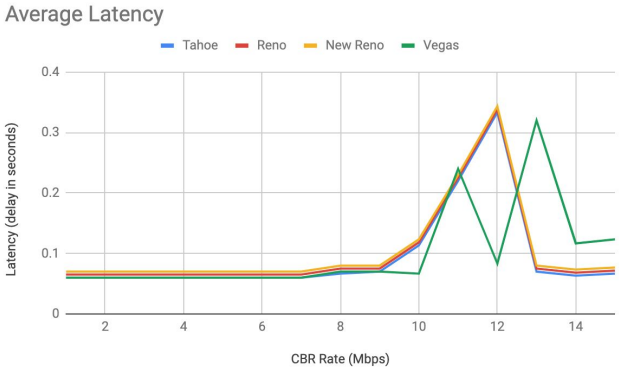

Figure 2.2 Average latency for Experiment 1.

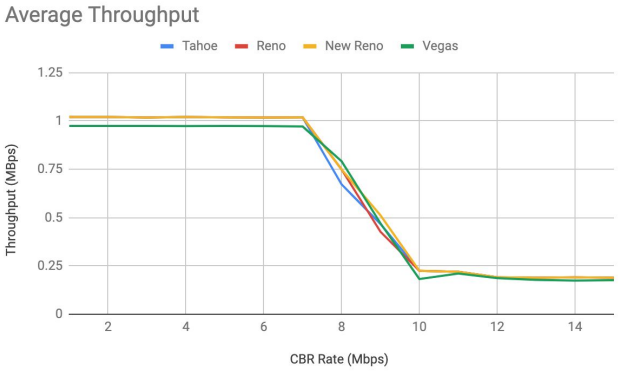

Figure 2.3 Average Throughput for Experiment 1.

Figure 2.2 shows that Vegas has an overall lower average latency than the other variants in somewhat congested networks. Once the network is completely congested, the other three variants actually perform better in terms of overall latency. This is likely because Vegas is able to drop fewer packets initially than the other variants due to its early congestion detection mechanism. Once it reduces its congestion window, however, it does not increase as quickly as the other variants and therefore the latency is increased in the network.

The results of the experiments show that the average throughput for all of the variants was relatively similar. All of the variants were able to utilize the entire bandwidth until the CBR reached 7 Mbps. After this point, the throughput fell to 0.25 MBps and remained constant thereafter. Reno, New Reno, and Tahoe all maximize throughput initially and have a higher average throughput than Vegas. TCP Tahoe has the lowest overall average throughput compared to Reno or New Reno, and this is likely because it does not implement fast recovery, where Reno and New Reno can reach maximum capacity faster during congestion control.

IV. EXPERIMENT TWO

The purpose of Experiment 2 is to determine if the TCP variants are fair to each other, that is, if they can share bandwidth evenly. To test this, we copied the network setup from Experiment 1 but added a second TCP stream with a source from N5 to N6 as shown in Figure 1. We tested the following four combinations of variants: Reno/Reno, NewReno/Reno, Vegas/Vegas, NewReno/Vegas. Each TCP flow transmitted packets of the same size, and we varied the CBR flow identically to Experiment 1. This time we varied the start time of both TCP flows such that one starts before the other or they start at the same time. We again plotted the drop rate, latency, and throughput for each combination and computed the averages of these values.

In the first combination of two TCP Reno variants, the flows behaved normally when the CBR rate was below 4 Mbps as shown in Figure 3.1. After this point, the total throughput decreased quickly from 1.02 MBps to .02 MBps for one flow and .06 Mbps for the other when the CBR rate was 10 Mbps. The average throughput of

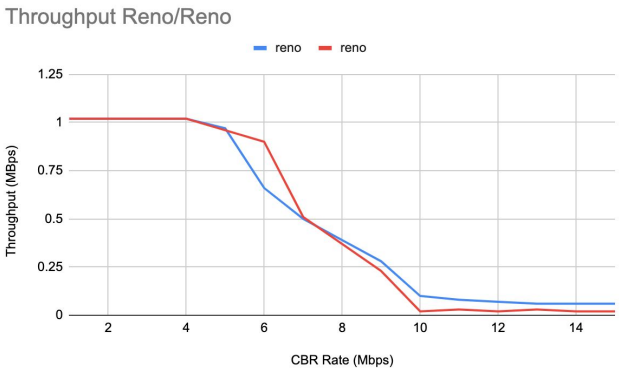

Figure 3.1 Throughput of two TCP Reno variants competing for bandwidth on the same link.

each flow was .483 ± .004 Mbps. The two flows oscillate between higher throughputs because one must receive the duplicate ACK first and decrease its congestion window. During this time, the other can transmit normally until it too receives a duplicate ACK and decreases its own window. By this time, the first flow has started to transmit at a higher flow. The two streams are fair to each other in congested and non-congested networks.

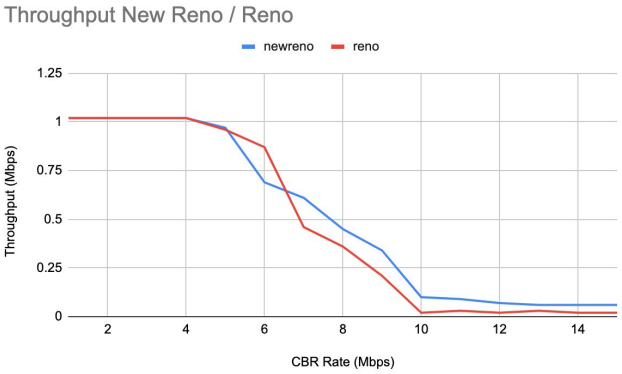

Figure 3.2 Throughput of TCP Reno and New Reno variants competing for bandwidth on the same link

Figure 3.2 shows the second combination of variants tested for this experiment. Here we see that New Reno actually hogs more bandwidth on average (.51 Mbps) than Reno (.47 Mbps). This is most likely because New Reno handles congestion in a more greedy behavior by only waiting for one duplicate ACK before retransmitting packets whereas Reno waits for three duplicate ACKs. In a congested network, New Reno

and Reno are not exactly fair in terms of sharing bandwidth.

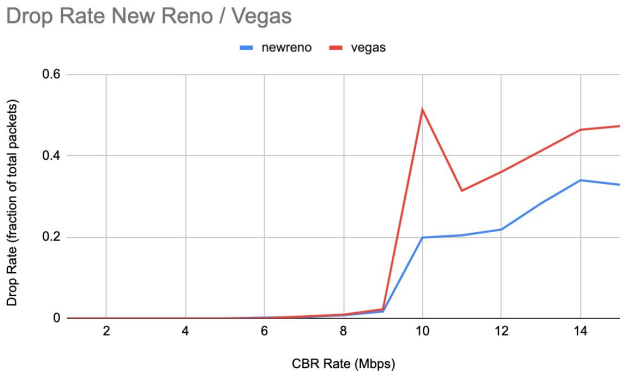

Figure 3.3 Drop rate of New Reno and Vegas on the same network across varying CBR rates.

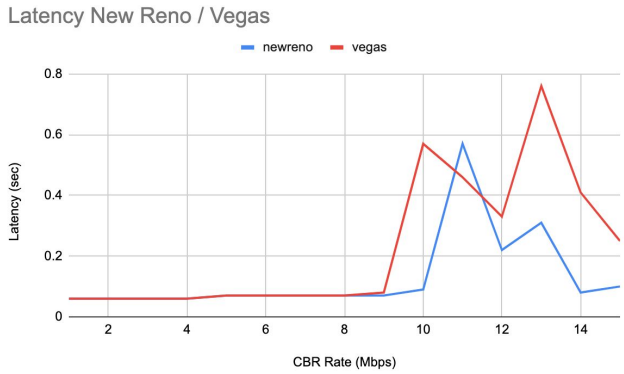

Figure 3.4 Latency of New Reno and Vegas on the same network across varying CBR Rates.

Of the four combinations, New Reno and Vegas shared the bandwidth the least fairly. As shown through Figures 3.3 and 3.4, Vegas has a higher drop rate and latency, and it also has a lower throughput. Its average drop rate was 17% with a peak of 51% in a congested network. New Reno had an average of just 10% with a peak at 34%. Vegas' average latency was nearly twice as long as New Reno's, with an average of .23s compared to New Reno's .13s. The average throughput of TCP Vegas was .45 Mbps compared to .50 for New Reno. In our experiment, the two variants were not fair at all in sharing the bandwidth. This is because, as explained in the results for Experiment 1, Vegas can detect congestion from RTT instead of packet loss. Therefore it decreases its congestion window before New Reno does, and then more packets are dropped as a result and the latency increases. Because New Reno retransmits so quickly by just waiting for one duplicate ACK, it is constantly sending a high number of packets even when the network is congested.

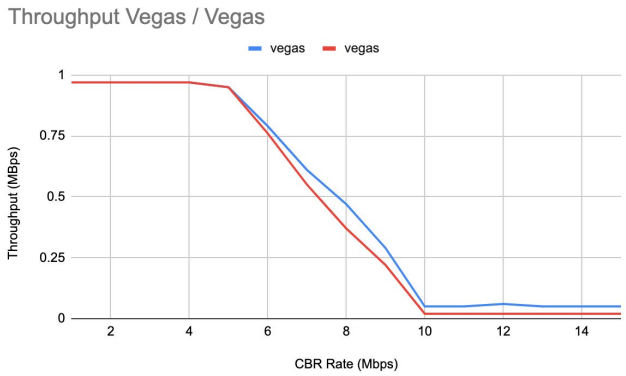

Figure 3.5 Throughput of two TCP Vegas streams competing for bandwidth on the same link.

In the final part of Experiment 2, we compared two TCP Vegas flows on the same network and analyzed how they behaved. In terms of throughput, the flows shared bandwidth fairly equally. Both flows began to decrease throughput at equal rates when congestion arose. One flow had a slight edge, averaging .48 Mbps compared to .46 Mbps. This is likely because one flow was able to detect congestion sooner and subsequently reduced its congestion window and never caught back up the the other flow's transmission rate. TCP Vegas behaves well with flows of its own variant type.

V. EXPERIMENT THREE

Experiment 3 studies the behavior of the network when the queuing algorithm of the TCP and UDP flows change. We studied two queuing algorithms: DropTail and Random Early Drop (RED). DropTail queues packets until the queue reaches maximum capacity, and any new packets after that are dropped until space is made in the queue for incoming traffic. RED uses predictive models to preemptively drop packets before the queue becomes full.

The network topology remained the same as Experiment 1 with a TCP flow from N1 to N4 and a CBR flow from N5 to N6. For each queueing algorithm, we ran tests using TCP Reno and TCP SACK as the TCP variant and we did not alter the CBR flow. We started the CBR flow 10 seconds after the TCP flow

began to ensure the TCP flow reaches a steady state. We also cut off the CBR flow after 40 seconds to see how the TCP flow recovered from the ease of congestion.

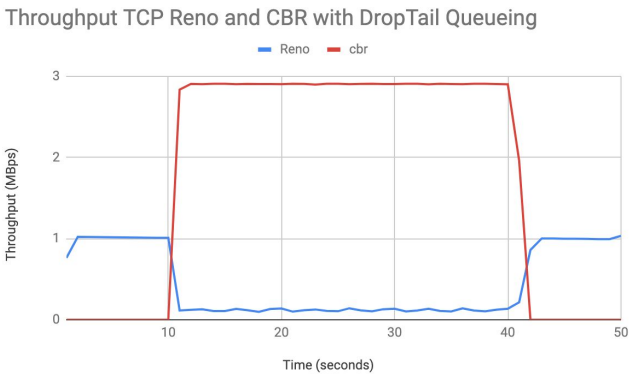

Figure 4.1 Reno and CBR throughput with DropTail queueing algorithm.

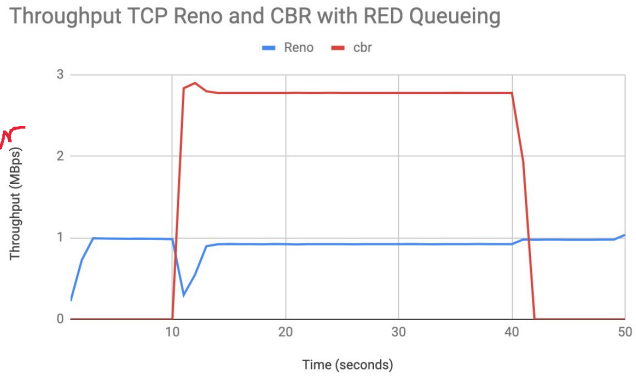

Figure 4.2 Reno and CBR throughput with RED queueing algorithm.

Figures 4.1 and 4.2 show the contrast between the throughput when DropTail is implemented versus RED. The average throughput for RED is .91 MBps while the average for DropTail is just .45 MBps. With TCP Reno, it is clear that both queueing algorithms do not provide fair bandwidth to each flow, however RED does a much better job than DropTail. In a similar fashion to TCP Reno, RED was more fair at sharing bandwidth between flows when TCP SACK was implemented, which can be seen in Figures 4.2 and 4.3. Although TCP SACK took slightly longer to recover from the introduced CBR flow. The flow also slightly increased back to full capacity once the CBR flow was dropped. Using DropTail, the queue was constantly full and all incoming packets were immediately dropped, therefore the throughput was very low.

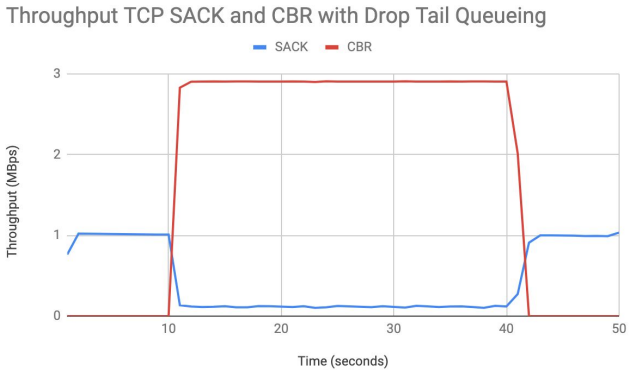

Figure 4.3 TCP SACK and CBR throughput with DropTail queueing algorithm.

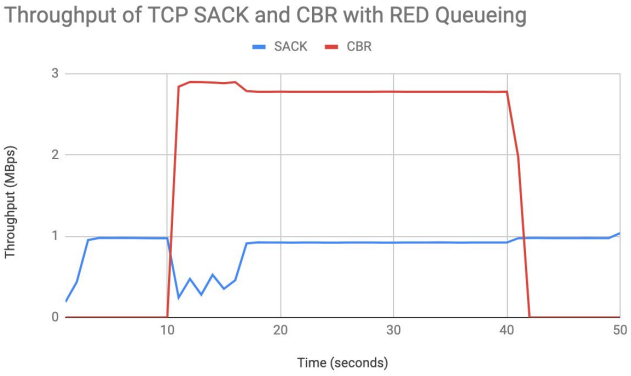

Figure 4.4 TCP SACK and CBR throughput with RED queueing algorithm.

The end-to-end latency when DropTail was implemented was higher than with RED. RED experienced almost no latency while DropTail had an average of less than .1 seconds average. This is likely because DropTail enters a phase called retransmit synchronization once its queue becomes full. This causes a bursty behavior from the transmitting node on retransmission. RED drops packets in a more consistent manner using its predictive models, which is why there is lower latency overall.

We implemented our experiments with a smaller than default queue size. In our results, we found that RED had fewer packets dropped, lower latency, and higher throughput. We initially ran the experiment with the default queue size and found that DropTail actually has a higher overall throughput. Depending on the network configuration, RED queueing could be a good

idea to implement while dealing with SACK. In the normal environment, DropTail might perform better. Our results do not strictly show this, but other research does indicate this behavior is true [3].

In general, the TCP flows do not react well to the introduction of CBR flows. CBR transmits blindly in the network at a constant rate while TCP reacts to congestion in the network. This means when CBR begins transmitting, TCP must decrease its congestion window because it experiences data loss. When the CBR flow stops, TCP is able to recover over time.

VI. CONCLUSION

The Internet is built of tens of thousands of networks which link together to connect billions of hosts. It is the job of the host OS to implement data transfer protocols like TCP, and each may do so differently. It is important that these different TCP variants work correctly, efficiently, and fairly. This paper tested the efficiency and behavior of four TCP variants in Experiment 1, their fairness in Experiment 2, and the influence of two queueing algorithms, RED and DropTail, in Experiment 3.

We found that in congested networks, TCP Vegas had the least latency and the lowest drop rate. The key takeaway from Experiment 2 was that TCP New Reno did not behave fairly with other TCP variants. This was obvious by its much higher use of the shared bandwidth. Finally, in Experiment 3 we concluded that RED queueing allows for greater use of the available bandwidth.

The different TCP variant implementations may cause implications on real systems. If two hosts exist in a network where one runs Vegas and another uses New Reno, the one on Vegas may experience slower connections because New Reno might hog the bandwidth. It is important that all hosts are sharing bandwidth equally if the users are paying the same amount for it.

Future studies may wish to study what the behavior of TCP variants is on larger networks, say with four or more variants. Also, there exist other TCP variants such as Compound and Cubic which are implemented by Microsoft and Linux respectively. Research indicates that Cubic may not be entirely TCP-Friendly [3], and more experiments could be performed in NS to prove that.

VII. REFERENCES

[1] Fall, Kevin, and Sally Floyd. “Simulation-Based Comparisons of Tahoe, Reno and SACK TCP.” *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, 1996, pp. 5–21., doi:10.1145/235160.235162.

[2] Chung, Jae, and Mark Claypool. *NS by Example*, Worcester Polytechnic Institute, nile.wpi.edu/NS/.

[3] Eva, N. Obinna, and L. G. Kabari. “Comparative Analysis of Drop Tail, Red and NLRED Congestion Control Algorithms Using NS2 Simulator.” *International Journal of Scientific and Research Publications (IJSRP)*, vol. 8, no. 8, 2018, doi:10.29322/ijsrp.8.8.2018.p8069.