Analysis of TCP Reno, New Reno, Tahoe, Vegas, and Sacks Using NS-2

Will Tome
Northeastern University
willt@ccs.neu.edu

Gary Soeller
Northeastern University
soelgary@ccs.neu.edu

***Abstract*—TCP is one of the original transport protocols that was built for the Internet. At the time when it was created, the developers did not have any idea how big the Internet was going to be, so they did not originally design TCP to handle congestion very well. Over time newer implementations of TCP were developed to better handle congestion and maximize the amount of throughput each client can get. Each of these different implementations use different algorithms for their sliding window, back off, and starting mechanisms. The differences in these algorithms are what decide how fast each implementation can send data and how fair they are to other hosts trying to send data over the same network.**

Our goal is to analyze the differences in five different TCP implementations: Tahoe, Reno, New Reno, Vegas and Sack. Through our analysis, we find out which implementations have the best throughput, the most dropped packets, and are most fair to other hosts on the network.

I. Introduction

TCP(Transmission Control Protocol) is a transport layer protocol designed to deliver packets from one host to another on a network. Each TCP implementation over the years was designed differently with the intent of being more efficient, so it would minimize the number of packets dropped while still being as fair as possible to its neighbors.

In this investigation we ran 3 experiments to gather data to make conclusions about the different implementations of TCP. The specific implementations we looked at were Reno, New Reno, Tahoe, Vegas, and Sack. In Section 2 we will explain the methodology we used to gather data and will go in depth for the 3 experiments. In section 3 we will talk about the results we found for each experiment and in section 4 we will conclude our findings and analysis.

II. Methodology

To conduct experiments, we used The Network Simulator (NS-2)[1] installed on CCIS Linux machines. Following an NS-2 template produced by Jae Chung and Mark Claypool[3], we constructed the assigned topology depicted in Figure 1.

The template file provided some extra code to support The Network Animator (NAM)[2]. While NAM was not used to conduct any formal results it served as a helpful tool to predict the outcome of raw data generated graphs. The template included the code to properly open, record and close a trace of the traffic flow. We analyzed the remaining code to ensure that it would preform the traffic flow as desired and specified. The traces were where packet data was stored once a test was run.

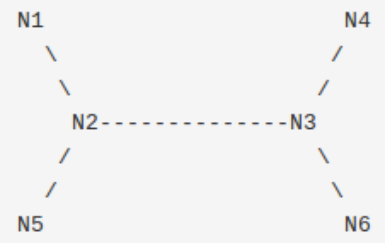

Fig. 1. Node topology

We processed the trace data with a python script to provide a more structured dataset that could be plotted by gnuplot.

Between each node is a link defined with $10Mb$ of bandwidth and $10ms$ of delay. Simulations consisted of a TCP flow between two nodes in the network with a Constant Bit Rate (CBR) flow somewhere along the path to create congestion on the link. The CBR was simply a UDP stream between two nodes. Most experiments were conducted by allowing the CBR flow a 0.5 second head start on TCP. By starting CBR first, we were able to see more TCP behavior as it had to ramp up in the face of contention.

A. Experiment 1

Experiment 1, was conducted using the topology in Figure 1 with a steady CBR stream from N2 to N3 and a TCP stream from N1 to N4. Starting with a CBR size of 1Mb, we increased the size of the CBR size incrementally up to the bandwidth of the link, 10Mb/s. The total experiment time was 5 seconds. The CBR flow was started at 0.1 seconds and TCP started at 0.5 seconds. The simulation ran uninterrupted until 4.0 seconds when TCP stopped followed by CBR at 4.5 seconds. This was observed to be sufficient time for TCP to preform its slow start and observe a back off if the queue overflowed. Using the same parameters, the experiment was run with four variants of TCP, Tahoe, Reno, NewReno, and Vegas. The data was recorded on how these TCP variations compared to each other in throughput, dropped packets and latency. Specifically we were interested in how each handled the different degrees of congestion caused by the CBR flow. The drop tail queuing algorithm was used on all links. In a drop tail queue, when the queue filled up it would drop packets that overflowed the queue.

B. Experiment 2

This experiment builds off of experiment 1 by adding an additional TCP flow from N5 to N6 (see Figure 1). In this case, not only was the size of the CBR flow changed but the experiment inherently had more congestion between N2 and N3 due to the addition TCP flow. Experiment 2 followed the same time profile as experiment 1. Both TCP flows started and ended at the same time and with the same relation to the start and end of the CBR flow. Again, similar to experiment 1, the drop tail queuing algorithm was used. In the preformed tests, different variants were put against each other to determine how they would behave in the face of the increased congestion and also how fair they would be to each other. Tests of Reno vs. Reno, NewReno vs. Reno, Vegas vs. Vegas, and NewReno vs. Vegas were preformed.

C. Experiment 3

The third and final experiment uses the same topology found in Figure 1. Experiment 3 was composed of a single TCP flow like in experiment 1 from N1 to N4 with a CBR flow between N5 and N6 to create congestion on the link. In this experiment, we perform tests with the link between nodes N2 and N3 using drop tail queuing algorithm and random early drop (RED) queuing algorithm. RED will use a more intelligent form of queue management by not simply dropping packets that overflow the queue but dropping packets within the queue to help it decrease and allow more flows through. This provides variability in how the queue is managed when it over flows. These tests are preformed using TCP Reno and SACK TCP implementations. The time profile for this experiment was still based on a 5 second duration however TCP was started before CBR to allow it time to warm up. TCP is started at 0.1 seconds, CBR started at 1 second. CBR then ended at 4.0 seconds and TCP carried on for another half second to come to a stop at 4.5 seconds. The CBR bandwidth was set to $9Mb$ for all tests. This was enough to create a bottleneck for TCP to react to without over saturating the links.

III. Results

A. Experiment 1

Recall that experiment 1 consisted of a TCP flow between two nodes and a CBR flow between two nodes along the TCP flows path. Through various tests it was determined that using a $9Mb$ CBR size provided the optimal conditions to test each variant of TCP. If we went below $9Mb$, we found that every packet was making it through, but if we went above, we found the queue would never empty if it ever got backed up.

The first TCP variant we tested was Tahoe. Tahoe ended up dropping 12 packets, which is the highest of all the experiments. It makes sense that Tahoe had the highest number of dropped packets when looking at the RTT of the recieved packets. Throughout the experiment, Tahoe is the only variant that had 3 high spikes for RTT. Although the spikes are not as high as the other variants, having 1 more is significat to the performance. The way Tahoe works is that when faced with congestion, it starts fast retransmit. The other variants use fast retransmit as well as other features to improve the efficiency. We can also see that the average throughput is more sporadic. The dips in throughput match the times when packets were dropped and dips in sequence numbers recieved in Figure 3.

The next TCP variant we analyzed was TCP Reno. Reno performed exactly the same as Tahoe at the beginning. The two started to differ after the first min in Figure 2. This difference is caused because Reno uses fast recovery when their is a dropped packet. As far as dropped packets go, Reno performed better than Tahoe by only dropping 11 packets. We can see in Figure 4 that Reno also has spikes in time where the RTT is much higher than usual. These are the times when throughput is at a minimum and packets are being dropped due to congestion.

Next we took a look at TCP New Reno. The main difference between Reno and New Reno is that it sends a packet for every ACK it recieves. Looking at Figure 2, we observe that New Reno never drops to a throughput as low as Reno. This confirms that New Reno does a better job of maintaining a higher throughput. Figure 3 confirms this as well because we can see that it takes less time to recieve each consecutive sequence number. Up to this point, TCP New Reno has the least number of dropped packets with having 10.

The final variant we analyzed was TCP Vegas. Of all of the variants, Vegas performed the best. It dropped 0 packets, had the most consistant throughput, and never had a big spike in RTT. It was able to successfully get all of the data to it's destination and it did it fast and without any problems.

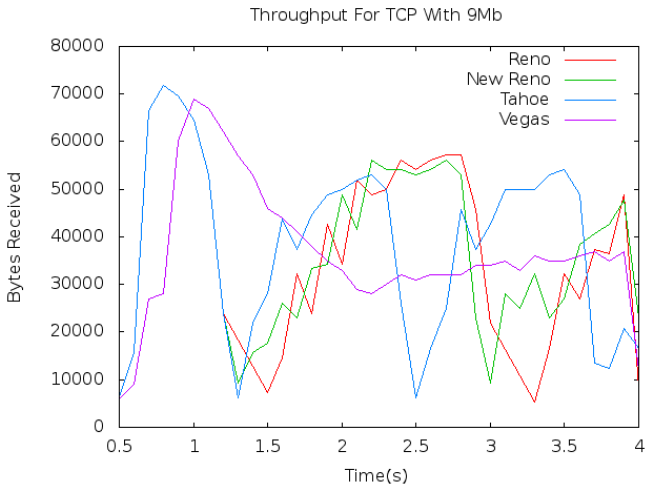

Fig. 2. All TCP Throughput with 9Mb CBR

B. Experiment 2

In testing the fairness between TCP variations, we found that most protocols behave fairly to themselves. When testing Reno vs. Reno and Vegas vs. Vegas, it can be observed that when started together they progress with the same behavior. In fact, they behave similar to the way they did when operation alone. Vegas found a steady throughput to sustain that reduced spikes causing dropped packets and exponential back off.

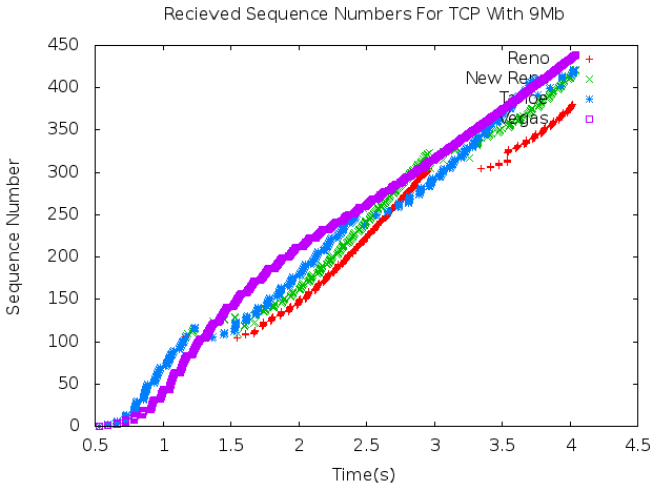

Fig. 3. All TCP Sequence numbers with 9Mb CBR

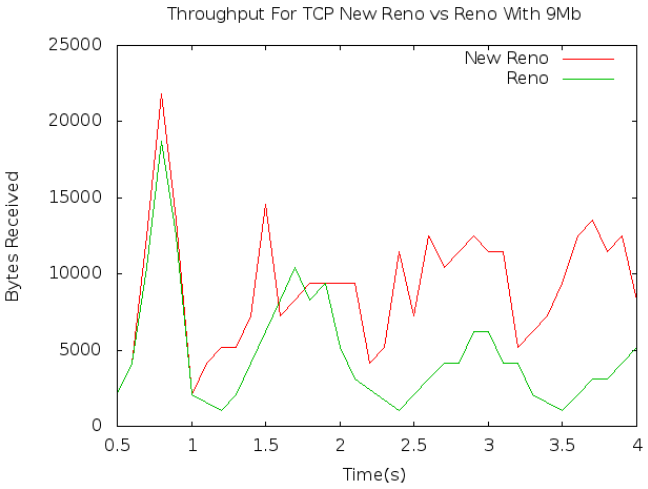

Fig. 5. Vegas Sequence with 9Mb CBR

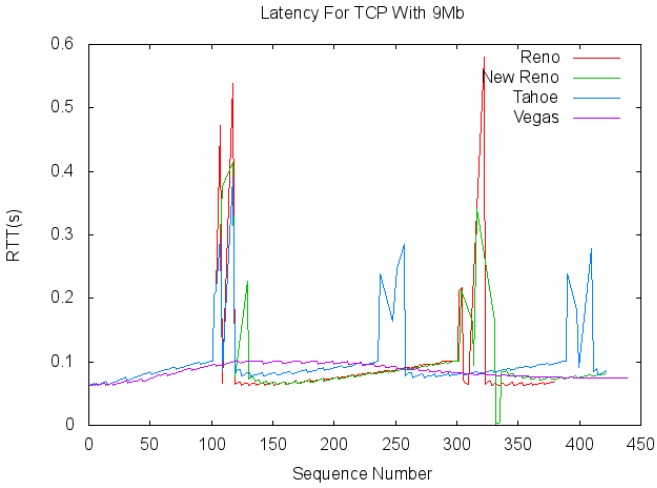

Fig. 4. All TCP Latencies with 9Mb CBR

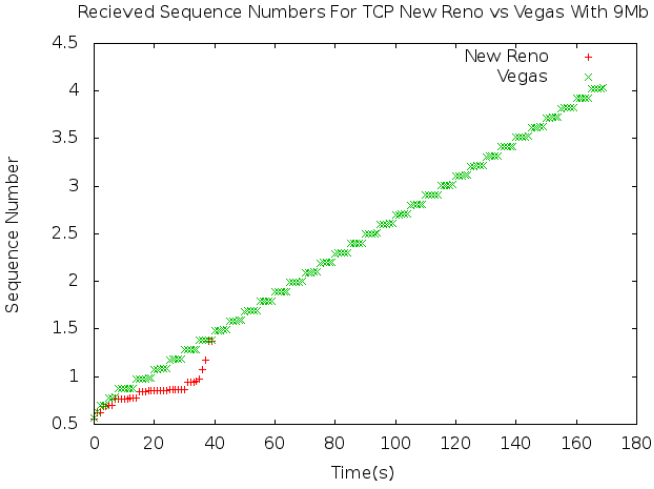

Fig. 6. New Reno vs. Vegas Sequence with 9Mb CBR

In the face of congestion Reno would back off a little and then continue again. When Reno and New Reno were put to the test against each other, New Reno started up a little bit faster than Reno. After the initial back off, New Reno then ramped up much faster and continued to take up more of the bandwidth in the network for the remainder of the test. We show the throughput of the of this matchup in Figure 5. The last experiment we ran was between New Reno and Vegas. Here we found that Vegas was not fair to New Reno about the amount of bandwidth it was using. As you can see in Figure 6, Vegas is consistent with the rate in which it sends its sequence numbers while New Reno backs off and doesnt really get the opportunity to ramp back up again. Figure 7 supports this result as well because it shows that after the initial ramp up, Vegas maintained a steady throughput while New Reno had difficulty getting going again.

C. *Experiment 3*

In the final experiment we experimented with TCP Reno and SACK for the drop tail and RED queueing algorithms. Our results show that the throughput is better for the drop tail algorithm. We can see this by taking a look at Figure 8 which shows the throughput for Reno with RED, Reno with tail, SACK with RED, and SACK with tail. Although the throughput is higher in drop tail, the latency is much better for RED. The reasoning behind this is that RED uses an algorithm to predict whether or not it will end up dropping a packet so it can drop packets early rather than having them sit in a queue for a while. With the drop tail algorithm packets spend much more time in a queue making the RTT significantly higher with a lower number of dropped packets.

Based on what we saw, there aren't really any combinations of queueing algorithms that work horribly with any version of TCP. It would be best if Reno didnt work with RED based

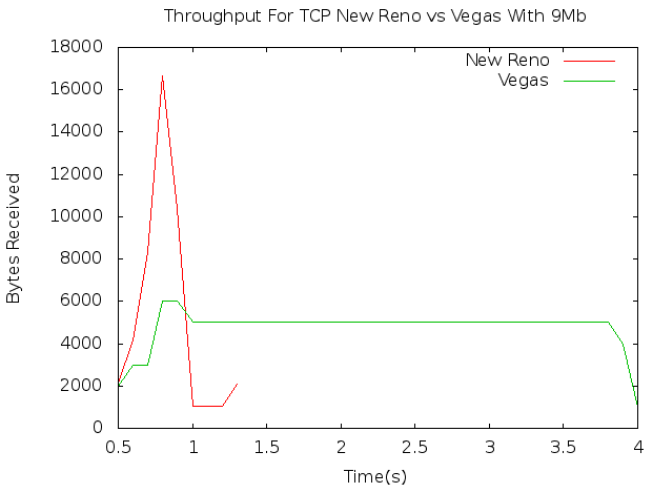

Fig. 7. New Reno vs. Vegas Throughput with 9Mb CBR

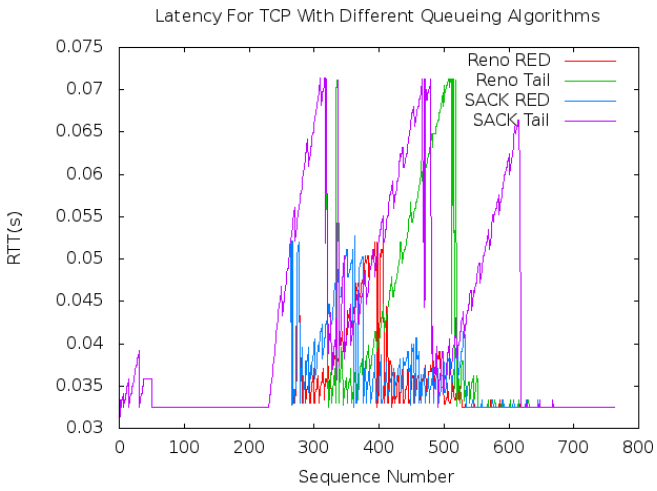

Fig. 9. Latency for TCP Reno and SACK for different Queueing Algorithms

on the nature of the two algorithms. With Reno not knowing which packets have been dropped from an ACK and RED dropping packets based on probability, the combination of the two makes the worst combination of them all. The best combination is SACK with RED. Sack is better than Reno because it knows which packets have made it and RED is the more efficient queueing algorithm because it is smarter about dropping packets. Since packets can be dropped out of order, SACK works well with it so it doesnt matter the order in which the packets were dropped.

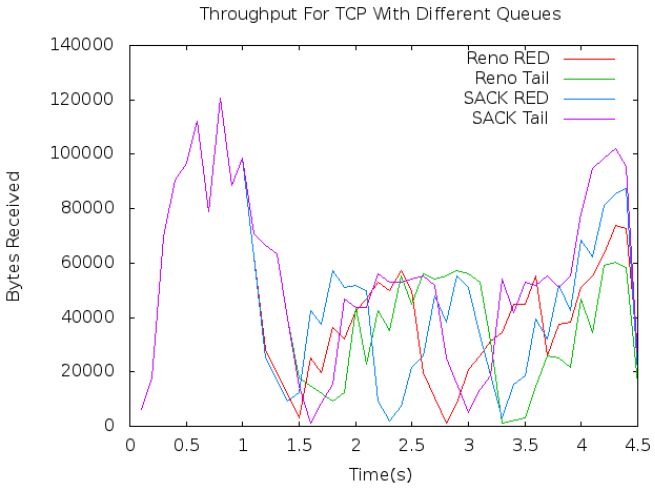

Fig. 8. Throughput for TCP Reno and SACK for different Queueing Algorithms

When the queuing mechanism is changed from droptail to red, Sack TCP exibits behavior of a quick short burst (about 1 second) of traffic before back off. Reno on the other hand, shows more sustained bursts of traffic lasting about 1.5 seconds.

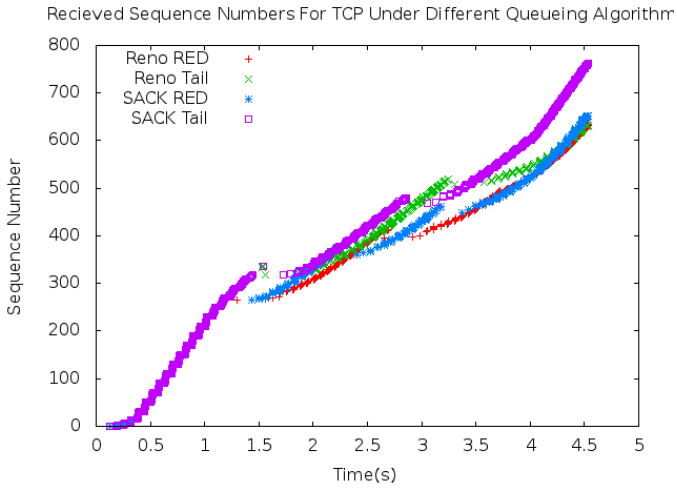

Fig. 10. Time Over Sequence Numbers for TCP Reno and SACK for different Queueing Algorithms

IV. Conclusion

The internet is designed not to treat traffic with preferential treatment rather provide best effort packet delivery. The OSI model's structure allows the flexibility to design multiple variations of a protocol to accomplish a task. Thus multiple versions of TCP exist to achive reliable transport over an unreliable network. Each version offers its advantages in how it handles congestion, latency and how to respond to dropped packets. However, the methods that are used to achieve better performance can sometimes hinder the performance of a competing protocol. It is unrealistic that all devices on the internet will use the same protocol or even be able to upgrade to the same protocol. In this paper we examined variants of the TCP protocol and analyzed them through 3 different experiments. Experiment 1 was to see how they perform by themselves with a CBR of 9Mb. Experiment 2 was to analyze how fair some variants were to others on the same network. Experiment 3 was

to analyze how different queueing algorithms implemented in the router change the behavior of each TCP implementation. For each experiment we ran simulations for 4 different TCP variants; TCP Reno, New Reno, Tahoe, and Vegas.

Overall, our results show that Vegas does a very good job of consistantly taking up the most bandwidth on the network without dropping many packets and having a very low RTT. These properties do not make it a nice protocol for other clients in the network. The next best variant in performance was TCP New Reno having less dropped packets than Reno and Tahoe, having a higher average throughput, and having a lower RTT on average. Of the next two variants, Reno outperformed Tahoe for having less dropped packets, a lower average RTT and higher average throughput. We also noticed that TCP Vegas does not play fair with other hosts in the network. Vegas does a very good job of getting it's data to the destination without getting any dropped packets.

REFERENCES

[1] http://www.isi.edu/nsnam/ns/

[2] http://www.isi.edu/nsnam/nam/

[3] Jae Chung and Mark Claypool, *NS By Example*, Worcester Polytechnic Institute