Performance Comparisons of Reno, New-Reno, Tahoe, and Vegas TCP

Dick Fickling, Eric Kelly, and Hardy Douglas

Northeastern University
360 Huntington Ave, Boston, MA 02115
{dfickl,ekelly,hdoug}@ccs.neu.edu

Abstract

In this paper, we use The Network Simulator (ns2) to compare four TCP variants – Reno, New-Reno, Tahoe, and Vegas – under three experiments with varying network conditions. The first experiment tests each variant as it competes on a network with a greedy UDP connection. We show that although throughput among variants is very similar, TCP Vegas has significantly less latency than the other variants, especially when the line is congested. The second experiment tests fairness – pitting the TCP variants against one another in an otherwise empty network. TCP Vegas performed the worst in these tests due to its non-greedy nature. Finally, we test two different network queuing approaches – random early detection and drop tail – and examine how the TCP variants perform with each approach. This experiment produced unexpected results, which will be discussed further below.

1. Introduction

The Transmission Control Protocol (TCP) was designed to add, among other things, reliability, flow and congestion control, and error detection to the networking stack. Since the protocol's 1974 introduction in "A Protocol for Packet Network Intercommunication," it has undergone a series of modifications to improve performance, mostly with respect to its congestion avoidance algorithm. Because TCP is so widely used and it is one of the core components of the internet today, it is important to understand how it works in a variety of network conditions. We primarily tested 4 flavors of TCP - Tahoe, Reno, New-Reno, and Vegas.

Tahoe, the original congestion avoidance algorithm, treats three duplicate ACKs as a timeout. After recognizing a timeout, Tahoe performs "fast retransmit", in which the slow start threshold is set to half of the current congestion window, the congestion window set to 1 * MSS, and the algorithm returns to slow start.

Reno was introduced to improve on the recovery time of Tahoe. When three duplicate ACKs are received, Reno halves the congestion window, sets the slow start threshold equal to the congestion window, retransmits the missing packet, waits for the entire window to be ACKed, and then returns to congestion avoidance.

New-Reno is a slight improvement over Reno, where it retransmits a dropped packet as soon as it gets a duplicate ack. This maintains high throughput even though a packet was dropped. New-Reno generally performs better than Reno.

Vegas uses a preemptive algorithm to avoid dropped packets by observing round-trip times and taking steps to avoid packet loss as RTT increases.

We simulated TCP performance under congestion, analyzing the latency, drop rate, and throughput as congestion increased. From this, we can understand how well TCP works for the most general case. We next simulated multiple TCP flows competing with the intent to determine fairness. It is important that one flow of data not dominate another flow, so we wanted to see if any flavor of TCP used more than its fair share of the bandwidth. And finally, we simulated the effect of different queue types – Drop Tail and RED - to analyze the effect various queuing algorithms had on overall throughput and latency. For this simulation, we used a fifth variety of TCP - SACK. SACK is a flavor of TCP which employs selective ACKs, a technique which improves TCP efficiency in the presence of dropped packets.

2. Methodology

Our experiments were conducted using the network simulator tool NS-2. We chose NS-2 because it is a very well-known networking simulator which is used in a variety of other research, so we can be reasonably sure that it is simulating network behavior correctly. Though NS-2 is extremely configurable, we primarily used the default values whenever we had the option. Our network topology consisted of 6 nodes and five links (figure 1). Each link had a queue type of drop tail, a delay of 10ms, and a bandwidth of 10mb/s. These values were chosen in order to give an approximation of the behavior of TCP over a network. There was constant UDP traffic between the two inner nodes (N2 and N3) to simulate network traffic. Between nodes N1 and N4 was the primary TCP link that we were analyzing, over which FTP traffic was simulated. In the numerous scenarios that we ran, we adjusted the traffic in the network (UDP constant send rate), the type of queues on each link, and flavors of TCP in order to have a surface understanding of how different forms of TCP interact with common network conditions.

```
N1                   N4
  \                 /
   \               /
    N2-------------N3
   /               \
  /                 \
N5                   N6
```

Figure 1: Network topology

3. Results

3a. TCP Variant Performance on Competitive Networks

Our analysis of throughput as a function of the CBR (network congestion) reveals that each variety of TCP is approximately equivalent in terms of average throughput. This makes sense, as we wouldn't want one flavor of TCP using more than its fair share of the available bandwidth. Tahoe does a little bit better than the other variants under very little network congestion, since the speed advantages of Reno and New-Reno don't apply unless packets are being dropped. Vegas does a little worse when there is little congestion, possibly because it does not slow start and exponentially flood the network. However, all of the variants do reasonably well in terms of overall throughput for all levels of congestion. We can also see a definite and understandable trend towards no throughput as the network gets more and more congested. As congestion increases, the differences in throughput also decrease. With less bandwidth to share, there is a smaller gap between throughput.

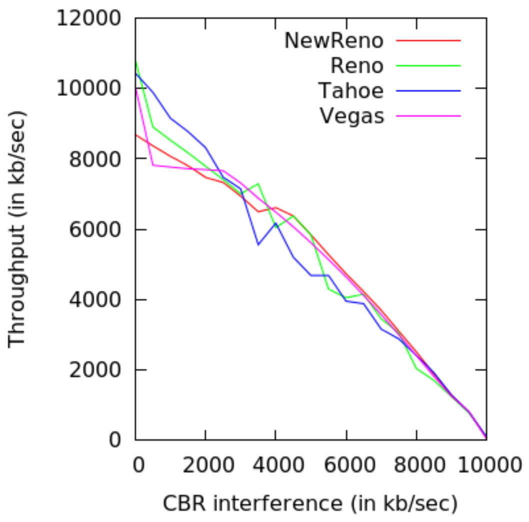

Figure 2: TCP Variant Throughput

Looking at the dropped packets graph (figure 3), we can see that Tahoe, Reno, and New-Reno all have equivalent packet loss. Between 0-80% congestion, the percentage loss remains below 3%. Between 80-100% congestion, packet loss skyrockets, as expected. What is more interesting is that Vegas performs incredibly well. Between 0-90% network congestion, TCP Vegas has *no* packet loss. This is because it uses an algorithm which attempts to prevent packet loss by analyzing round trip time (RTT). Vegas does not need to drop packets in order to adapt to network congestion, whereas dropped packets are an integral part of handling congestion in the other three algorithms.

The Latency vs. CBR graph (figure 4) shows some interesting results. Unsurprisingly, TCP Vegas does incredibly well - the preemptive algorithm it uses reduces latency so much that Vegas *always* does better than Tahoe, Reno, and New-Reno. Tahoe, Reno, and New-Reno on the other hand do universally poorly, jumping to around 80ms delay as soon as there is minimal network congestion. This is because Tahoe, Reno, and New-Reno detect congestion as packet loss. Thus, they experience far more timeouts in the process of recognizing and adapting to network congestion. There is no clear winner among those three variants when it comes to latency, since they all use that basic mechanism for managing congestion. However, the overall best TCP variant is clearly Vegas.

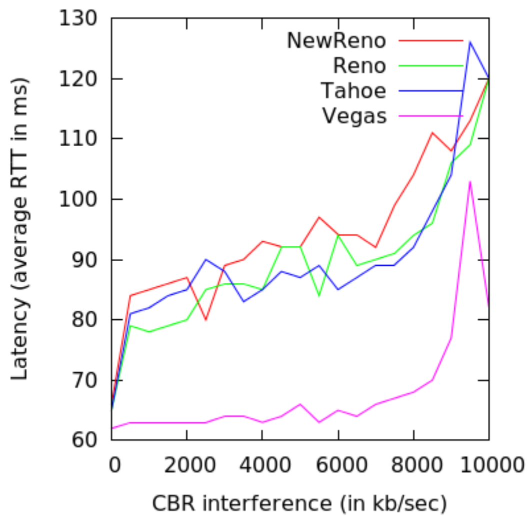

Figure 3: TCP Variant Latency

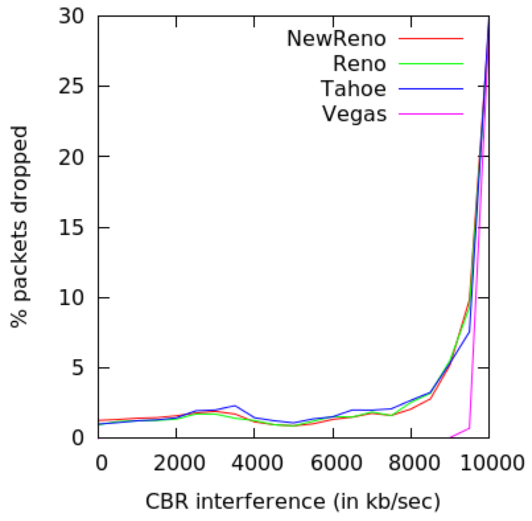

Figure 4: TCP Variant Packet Drops

3b. Fairness Between TCP Variants

We found that TCP Vegas plays extremely nicely with itself (Figure 5). Two Vegas flows had near identical drop rate, latency, and throughput over the entire range of network congestion. Also good news for Vegas - it has universally lower latency than a competing New-Reno TCP flow. However, bad news for Vegas - when pitted against a New-Reno flow, it gets overwhelmed in terms of throughput. This is because Vegas is way too nice. Vegas can react more quickly to congestion in the network, reducing its own send rate to keep a low latency and reduce network congestion. However, this frees up bandwidth for New-Reno, which gladly takes it. However, having two TCP varieties competing did not seem to significantly affect the percentage packet drop rate for either variety.

Two TCP Reno flows competing tend to have the same rate of dropped packets and latency - however, their throughput graph (Figures 6,7) reveals that when two Reno flows compete, one flow will dominate the other. This is surprising, because both flows are obviously using the same congestion algorithm. You would assume that it would be fair to itself first, the rest of the world second. However, one flow is consistently sending more data than the other flow. Although the dominating flow swaps as congestion increases, it is clear that two Reno flows do not equally share bandwidth. This could be because Reno is just incredibly slow at stabilizing on a fair bandwidth. Since Reno is a very aggressive flavor of TCP, it is too eager to grab bandwidth and that can result in one Reno flow using more than its fair share. Similarly, when a Reno flow competes with a New-Reno flow (Figure 8), throughput is not equal. New-Reno tends to use more than its fair share of the bandwidth, because New-Reno speeds up even faster than Reno when recovering from packet loss. When encountering a duplicate ACK, New-Reno immediately sends the missing packet, thus keeping a higher throughput than Reno, which must wait until it receives 3 duplicate ACKs.

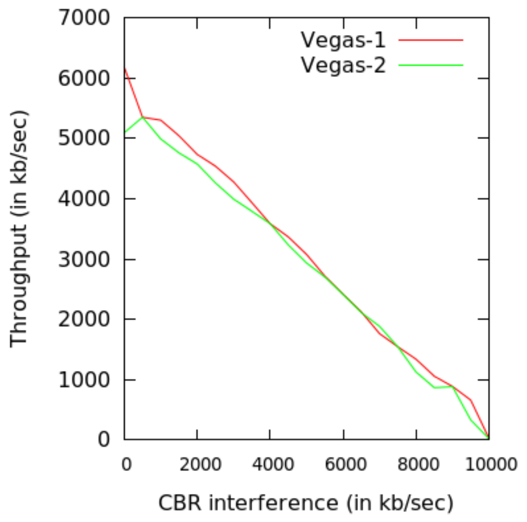

Figure 5: Vegas-Vegas Throughput

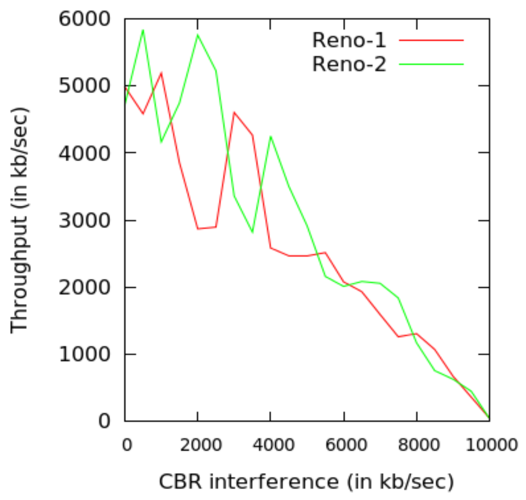

Figure 6: Reno-Reno Throughput

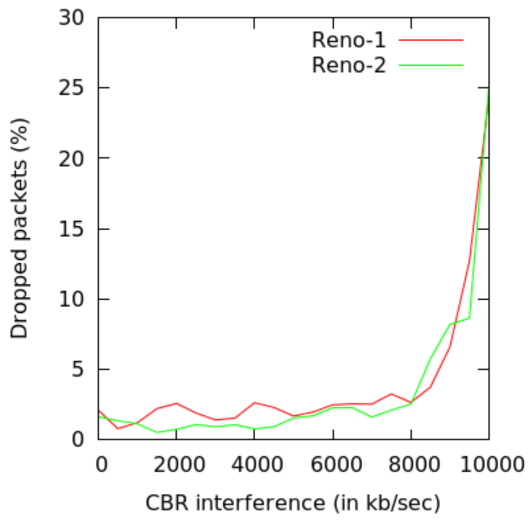

Figure 7: Reno-Reno Packet Drops

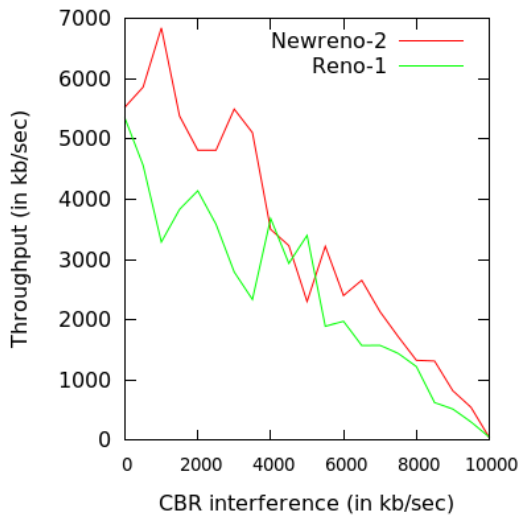

Figure 8: Reno-New-Reno Throughput

3b. Effects of Queuing Method Variations on TCP Performance

In our third experiment, we compared the performance of Reno and SACK when the router's queuing method is modified. We do see that when TCP packets are dropped and the flow backs off, the CBR throughput increases to fill the gap. This pattern is more pronounced in Drop Tail, so RED ends up causing a more "fair" flow.

The UDP flow enters the network at second 5 and causes immediate drops for TCP. TCP had been able to fully utilize the network, but with the introduction of the UDP traffic, had to enter congestion avoidance. Both TCP Reno and TCP SACK drop to 0 throughput when the CBR flow is created. They quickly bounce back to utilize as much of the line as possible, but the CBR flow overwhelms the network.

Each queuing algorithm causes drops, which in turn causes an increase in latency for both varieties of TCP. The amount of the latency does not seem to depend on the queuing algorithm. However, because RED causes more drops, the overall latency for Drop Tail is slightly better. That said, RED seems to handle bursty traffic more elegantly for both flows.

Using the SACK variant with RED queuing results in 10 dropped TCP packets between the start of the CBR flow and the end of the experiment. During this same period, Drop Tail queuing will only drop 6 TCP packets. Why? SACK has a higher throughput. Reno will wait for 3 duplicate ACKS to resend a packet, so SACK is faster. Because SACK is faster, it experiences more drops.

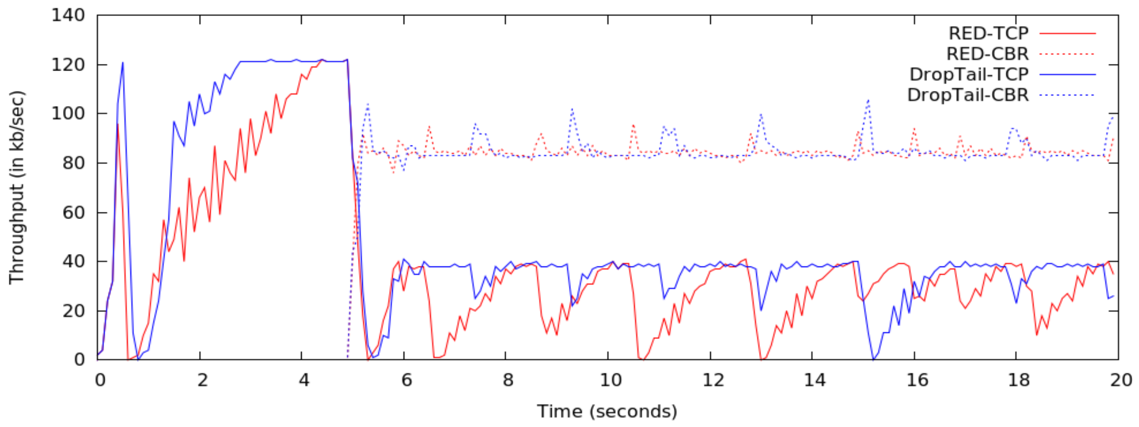

Figure 9: TCP Reno vs UDP Throughput Over Time

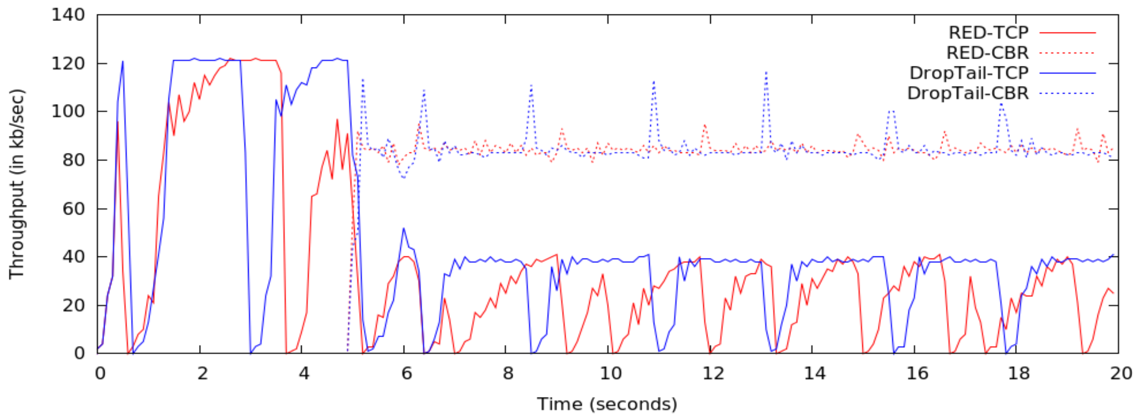

Figure 10: TCP Sack vs UDP Throughput Over Time

4. Conclusion

In this paper, we showed the results of a series of experiments in which we tested the real-world performance of TCP variants Tahoe, Reno, Vegas, and New Reno. Our first experiment confirms that, were Vegas to be deployed ubiquitously, latency across the Internet would drop considerably. Unfortunately, experiment two shows that the existing TCP variants are much more greedy than Vegas, and so any attempted rollout would cause Vegas users to have extremely slow, high-latency connections until the rollout was complete (which, given the scale of the modern Internet, would never happen). The third experiment showed, unexpectedly, that for the TCP variants tested, drop tail queuing performs better than RED. The authors discussed this anomaly with other researchers and discovered that such unexpected results were consistent across numerous experiments. Further research is necessary into why this anomaly occurred.

TCP Vegas is clearly the best TCP variant in terms of throughput and latency - except when used with any other variant. Therefore, if you are deploying a self-contained network, we would recommend using TCP Vegas to communicate internally, as you would see low latency and high throughput over the computers in the local network.

Due to these simulations, we now have a more thorough understanding of the relationships between various TCP variants - how they interact under congestion, how they interact with each other, and how they interact with queuing mechanisms. More importantly, we now know that only TCP Vegas can share bandwidth fairly, but because all other TCP variants will take advantage of TCP Vegas, we cannot recommend that it be used outside of a local completely controlled network.

Future studies should investigate other variants of TCP that we did not test - specifically TCP Compound and TCP Cubic. Additionally, more research needs to be focused at the effects of queuing algorithms on TCP flows, since our data had some elements of uncertainty.