Performance Analysis of TCP Variants

Daniel Piet
pietdaiel@gmail.com

Andrew Barba
abarba@ccs.neu.edu

1. INTRODUCTION

The Transmission Control Protocol (TCP) has come a long way since it was first introduced in 1974 by Vint Cerf and Bob Kahn. Although reliable, it struggled to perform well under congested networks which sparked the introduction of many different variants of TCP. In this paper we look to understand congestion and how different variants of the TCP perform under congestion. Understanding congestion is important because protocols behave differently under varying levels of network activity. To build a reliable network these subtle behaviors within the various TCP agents must be understood and analyzed.

Fairness is akin in this regard, as the interaction between various TCP agents can cause detrimental effects to network stability if their nature is misunderstood or unknown. Investigating the behavior of protocols in simulators allows for suppositions to be crafted with supporting evidence which can be tested on the open networks. Network simulators allow for hypothesis to be formulated over the observed controlled conditions. This is greatly beneficial as it aids in understanding the nuances of complicated behavior while being able to test its causal relations to the conditions of the network.

We have conducted three different experiments regarding TCP and congested networks. The experiments look to analyze overall performance (such as drop rate, throughput and latency), fairness between TCP variants, and the influence of packet queuing between different variants.

2. METHODOLOGY

Our experiments relied on the NS-2 Network Simulator; a highly used and credited system that can accurately simulate "TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks". It was supported by DARPA, Xerox and others in 1995 and continues to be supported today.

We used NS-2 to create a basic network topology that consisted of some number of TCP streams and some amount of noise. Each experiment differed in how the streams and noise were configured but for each simulation we generated trace file dumps that allowed us to analyze all packets sent through the simulated network. These files were then parsed into python objects for easy access and data manipulation. The objects were merely wrappers for the datagrams within the trace file; the packets were then encapsulated within a simulation object which allowed for further abstraction away from the trace dumps. From here we constructed functions to determine the throughput, latency, and drop rate of various packet streams. Throughput was calculated as bytes per second, latency was calculated in seconds, and the drop rate was determined by the total packets lost over time.

With this data we generated multiple graphs for each experiment that overlaid the some of the key differences. The graphs were built by accumulating statistics within a delta amount of seconds. For drop rate, the delta was set to a relatively high number to match the relatively low drop rates, this was 0.5 seconds. Latency and throughput required a smaller delta and were set between 0.1 and 0.2 seconds. For the goals of the experiment this was sufficient but if we were to continue further analysis we would rectify this approach in various ways.

3. EXPERIMENT 1

In Experiment 1 we performed tests that analyzed the performance of different TCP variants in the presence of a Constant Bit Rate flow. The network topology was set up as follows.

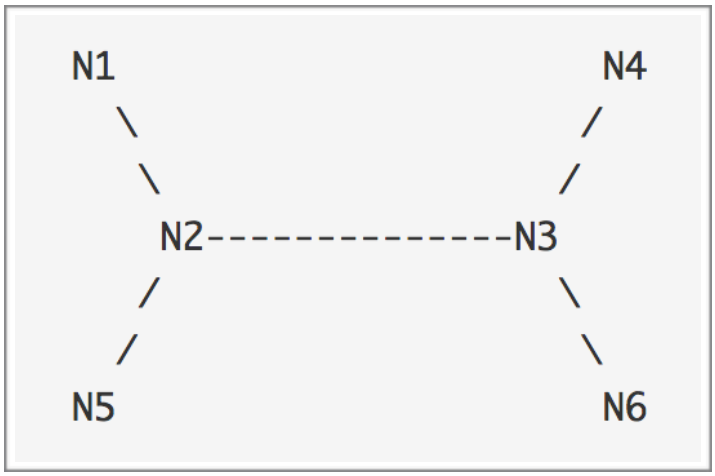

Figure 1. Network Topology

We sent a variable rate CBR from N2 to N3. We then proceeded by sending a TCP stream from N1 to N4. We used this set up against a varied array of CBR flows starting from 1mb/s up until 9mb/s. This set of CBR flow rates was tested against TCP Reno, Tahoe, NewReno, and Vegas. The NS-2 trace files were then parsed via the python script and graphed via matplotlib. For each tcp variant we graphed the drop rate, throughput, and latency over time. This allowed us to graph each CBR test on the same page to ease the analysis process. In doing so, some interesting results were obtained.

We found that at low CBR the four tcp variants behaved the same in terms of throughput and drop rate. This can be seen in Figure 2 below.

As can be seen, the behavior of the four variants at CBR's below 8 mb/s is nearly identical when comparing drop rate and throughput. After the network reaches ~80% utilization we see the variants beginning to behave differently. Latency was interesting here because Vegas had different latency patterns when compared to the other three variants. As can be seen from the graphs, at high network utilization, Reno, Tahoe and NewReno behave similarly in terms of latency. There is a increase in latency at the start of the CBR flow around 1 seconds which then slowly diminishes. While Vegas did not seem to have this problem. While it would be nice if a single TCP variant was able to get higher average throughput it seems as though this is not the case. At 8mb/s CBR on a 10mb pip Vegas was able to maintain a surprisingly high average at .72 Mb/s throughput. A full 1Mb/s over NewReno. Tahoe, Reno, and NewReno behave similarly with NewReno beating the two previous incarnations in all brackets. From the data analysis the lowest average latency was reported as Tahoe. This was surprising to me and may be a flaw in the data. I'd like to continue this with more expansive testing to truly determine the lowest latency TCP variant, both in simulation and on the live network. TCP Vegas had the fewest drop rates within the experiment which can be seen from figure 5.

To determine if there is an overall "Best" TCP variant we must consider the state of the network. While Vegas was able to beat the three other variants in many ways, its poor behavior when the network is at high utilization makes it not as desirable as NewReno when the network is highly congested. Its latency is also more than New Reno so given a situation where latency is a priority Vegas may not be the "best" choice.

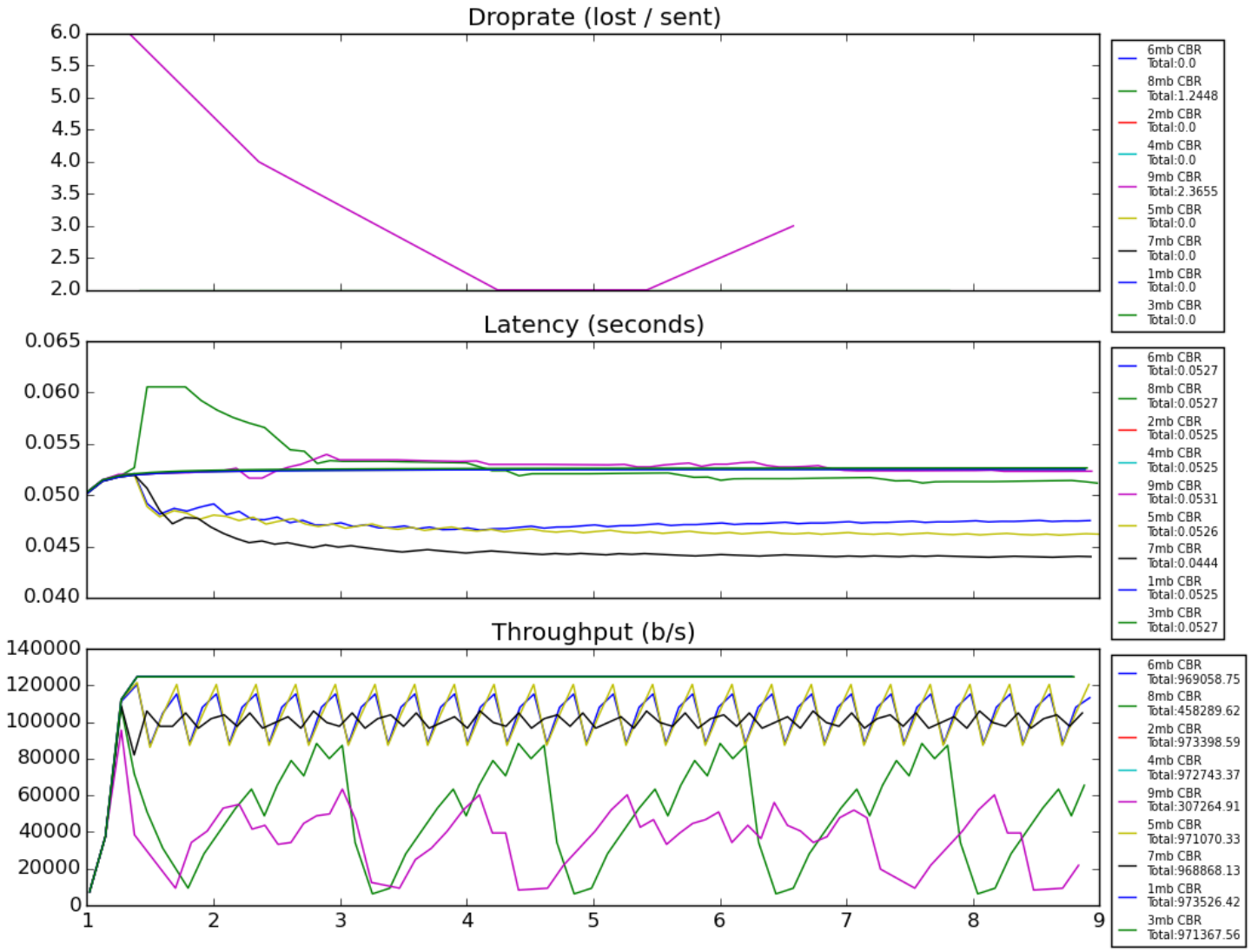

Figure 2. TCP Reno experiment 1.

4. EXPERIMENT 2

In experiment 2 we conducted a set of experiments that analyze the fairness between different TCP variants. The procedure was similar to that in experiment one. The only difference is the graphs plotted two agent's throughput, drop rate, and latency over time. A cleaned example set of result can be seen in Figure 3 below. It would be amazing if all TCP variants behaved fair over the internet, that is, each TCP variant utilizes it's fair share of bandwidth. Our experiments show this is not the case. The network topology used was the same as experiment 1 except that there was an additional TCP flow from N5 to N6. This allowed us to test the following pairs of TCP variants: Reno v Reno, NewReno v Reno, Vegas v Vegas, NewReno v Vegas.

Some of the interesting behavior observed can be seen from our results by analyzing the graphs. For Reno vs Reno it is shown below that it has wildly frantic behavior at high CBRs. When comparing New Reno v Reno at high CBR it can be seen that New Reno dominates Reno. Yet, at lower CBRs an phasic behavior is observed that becomes stagnant. For Vegas v Vegas there is a similar phasic behavior except it is also stagnant at high CBRs. I found this result surprising and it shows that the implementation of Vegas is better at maintaining consistency at high CBRs. Another surprising result was at mid level CBRs one agent would dominate the other. This violates fairness and was also observed when comparing New Reno and Vegas with New Reno dominating Vegas.

The different combinations of variants are generally not fair to each other especially at specific network utilizations percentages. In the experiment it was shown that Vegas was unfair to itself against the 7mb CBR flow. Similar competitive behavior was observed with Reno Reno at high network utilization. While there was excessively competitive behavior it did not seem a persistent dominator was established. Though this experiment was interesting, the extent at which it was carried out could be expanded greatly. A larger network with more agents would provide more information on fairness, along with larger RTT times.

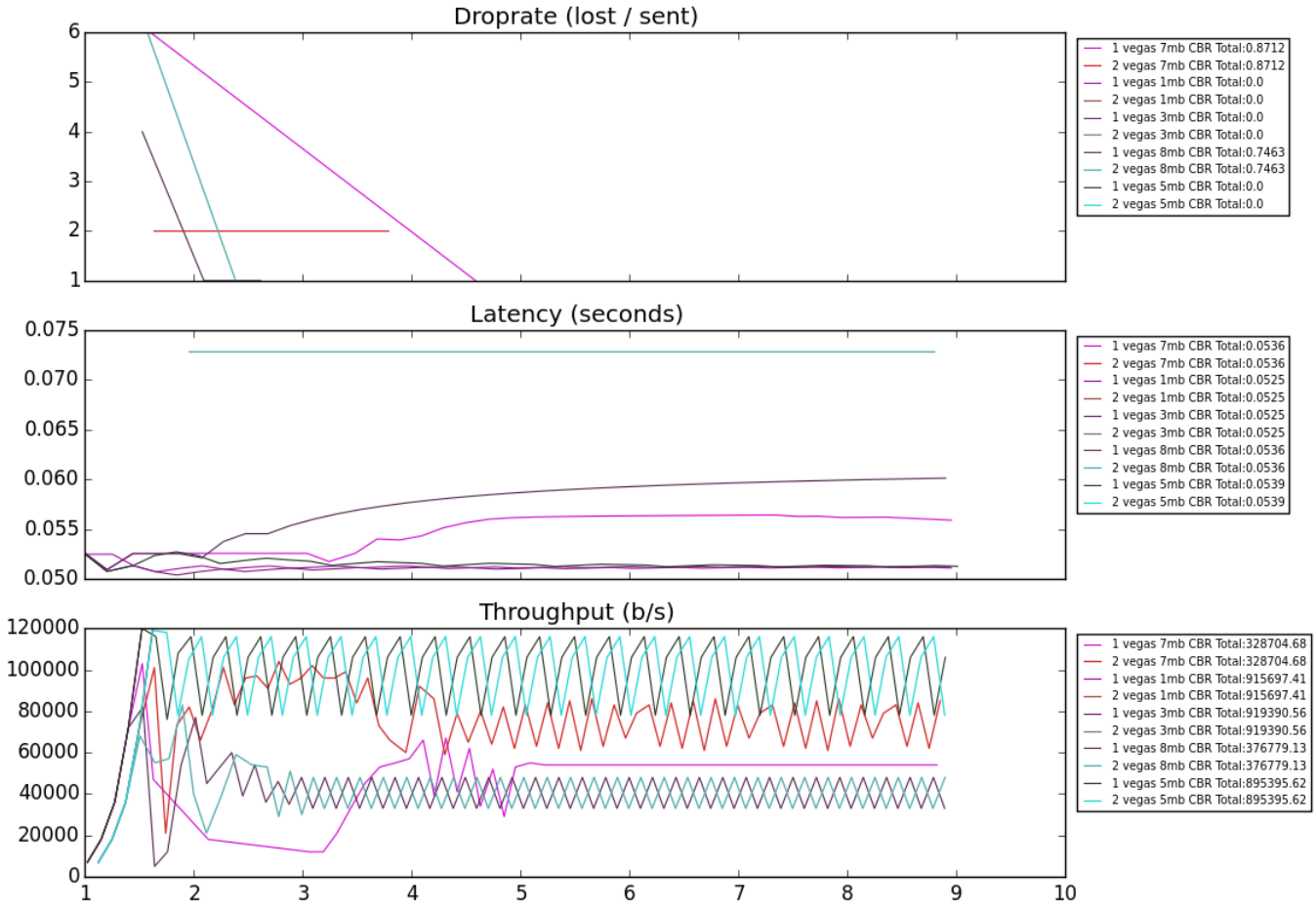

Figure 3. Vegas - Vegas experiment 2.

5. EXPERIMENT 3

In our final experiment we looked at two different types of packet queuing algorithms: Drop Tail and Random Early Detection (RED). Queue management algorithms are a necessary piece to any router or gateway; they bring fairness to the different types of senders and receivers by managing packet queue size and dropping packets when a sender is too aggressive. Drop Tail is the simpler of the two algorithms, literally dropping the "tail" end of the queue once it is full; it simply drops any packet that enters a full queue. RED is far more interesting as it attempts to fairly drop packets based on statistical probabilities.

Using the same topology from experiment 1, we simulated a single TCP stream from one end of the network to the other. This stream was allowed to warm up for 10 seconds until its throughput leveled out. At this time we started a constant 8Mb CBR flow that also ran from one end of the network to the other. The behavior TCP variants SACK and Reno can be seen below in figure 4. Even without knowing when the CBR flow was initialized, it becomes quite obvious in the graphs that something began creating network noise at exactly 10 seconds. In all cases, the TCP streams were consistent in throughput and leveled out in latency. Once the CBR flow began, packets began dropping and latency and throughput took a hit. End-to-end latency and throughput behaved very similarly in Drop Tail and RED when using TCP SACK. Latency declined steadily and throughput dropped drastically to 20,000 bytes per second, reaching a max of 80,000 bytes per second and then repeating. TCP Reno saw a steady increase in latency and better throughput utilization when using Drop Tail. Although RED matched Drop Tail in the best case, it was 50% worse in network utilization in the worst case.

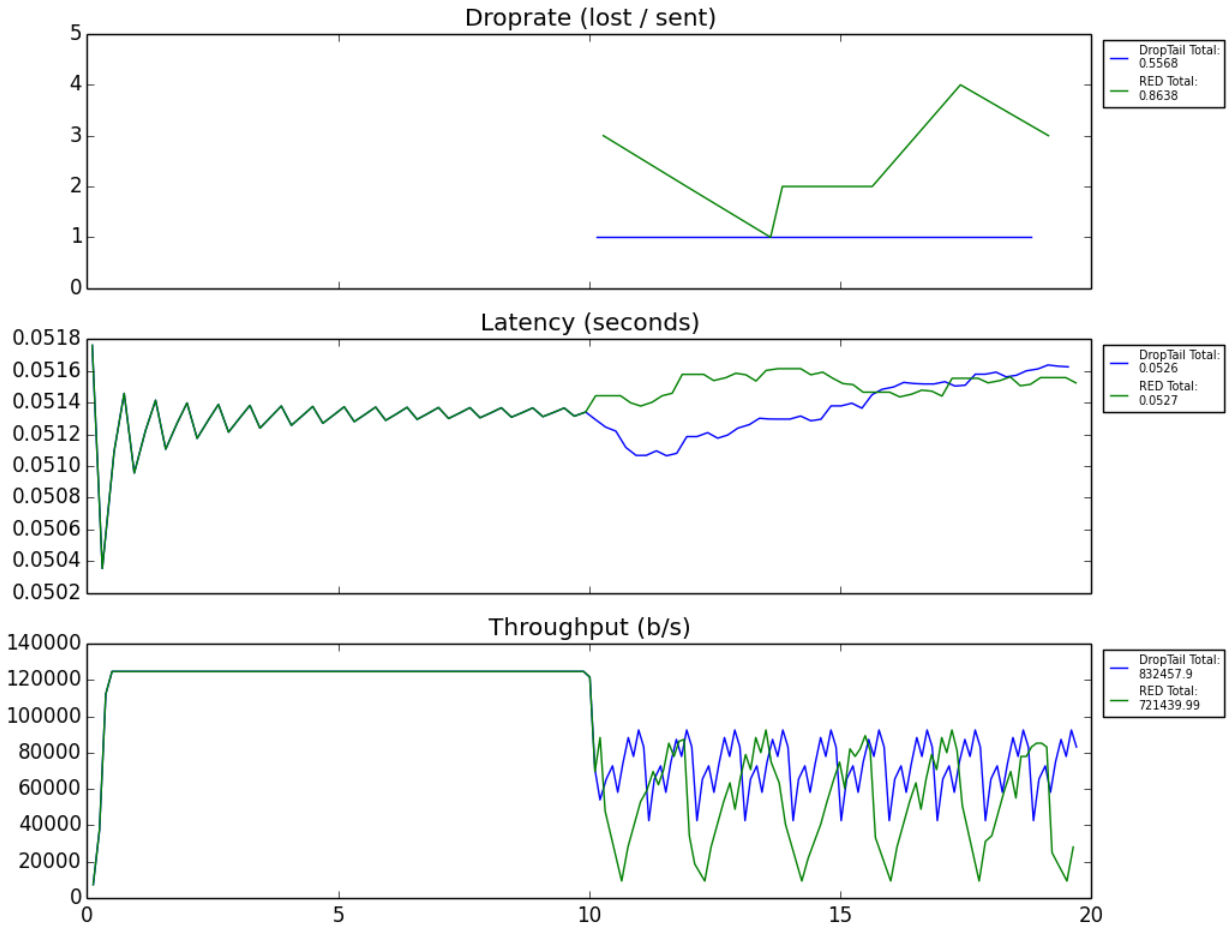

Figure 4. TCP Reno experiment 3.

6. CONCLUSION

TCP is an interesting protocol that has subtle implications when running network simulations with various configurations. The set of experiments carried out today has real world significance. For experiment one, understanding congestion allows for better protocol design to increase reliability. For experiment two, understanding fairness of TCP variants allows us to build better tools to optimize network utilization and ensure TCP variants are not clobbering other variants on the network. Finally, experiment 3 shows that when a compromise is necessary it is important to compromise fairly and in a manner that does not effect overall network performance. packet queuing algorithms are an easy way to do this throughout a network without leaving it up to the applications. The experiments lends itself to the conclusion that protocols should be used based upon previous knowledge of network conditions. Given the results from experiment one and two it can be seen that different protocols function better at high or low network utilization. In order to optimize network traffic some knowledge of the network conditions will lead to better efficiency. Some open questions that have arisen in the process of the experiment are, why is tcl a thing. Really, this language is ridiculous and I stand by the glorious Richard Stallman on this one. More serious questions are, how do more modern TCP variants--like Cubic--behave in the simulated conditions. Another avenue of future research is changing the RTT to be reasonably large to simulate traffic over high latency connections, and again in the other direction to simulate a data center type situation. Finally, another area of future research is to adjust the inner nodes to reorder the packets heavily to see how the different tcp variants behave under heavy reordering.