Nicholas Jones
Michelle Suk
CS 4700: Networking Fundamentals
Professor Choffnes
March 18, 2015

Performance Analysis of TCP Variants

Introduction

Since the introduction of TCP in 1974 (Microsoft Technet), many improvements have been made to the original algorithm. TCP variants were implemented in order to face the problems regarding the larger and more congested networks. This report presents research into the effectiveness and fairness of TCP variants under a variety of conditions including unresponsive data flows, competing tcp clients, and differing network queuing algorithms.

Methodology

The experiments discussed in this report were simulated using NS-2. This allowed us to run multiple simulations of the TCP variants and quickly. It also enabled us to control certain variables while randomizing others. All experiments were run on an H shaped network.

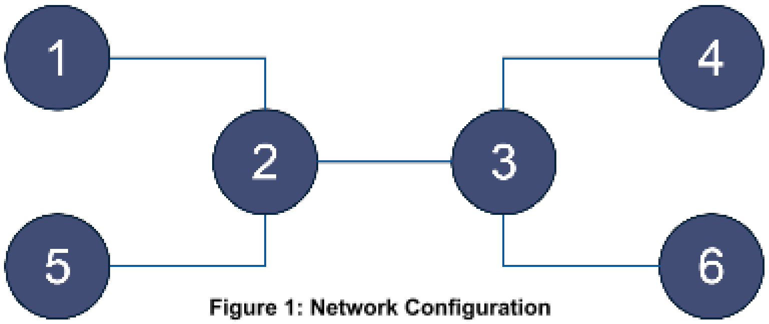


Figure 1: Network Configuration

To generate average results for each experiment, we ran experiments multiple times with randomized configurations. Some configurations randomized were start-time, CBR flow rate, and queue length; however, which properties to randomize were chosen on a experiment-by-experiment basis.

Simulation data from NS2 was obtained using the trace-all command. This output allowed post-analysis of the simulations, however, it was also verbose, recording events that didn't impact our simulation results. This presented two problems: trace-files took up GBs of space, and data write-time took up a majority of the run time. To reduce these issues and to limit our use of the undocumented, unreliable TCL, we ran NS2 as a subprocess under Python, catching the trace data from stdout and aggressively filtering it using regular expressions so that we only had to look at dropped packets and packets at the end-points. At the end of each iteration, we logged the results using the Pandas library, which allowed us to store our results to a CSV file after all simulation iterations completed. Pandas also provides integration with numpy and matplotlib, which we used to interpret and graph our results.

Experiment 1

The purpose of experiment 1 was to test how different TCP variants react to the introduction of a Constant Bit rate Flow (CBR). The network was configured by making a CBR flow from Node 2 to Node 3 and a TCP connection between Node 1 and Node 4. The TCP connection was started at the beginning of the simulation. After a random period (between X and Y), the CBR was turned on. This experiment was performed using the TCP variants Tahoe, Reno, NewReno, and Vegas. The test was also run against CBR Bandwidths from 0 to 10.5 Mbps with a step of 0.5 Mbps. For this experiment we randomized tcp start time, Node2->3 queue limit, and the CBR randomness using the random_ property. For each TCP/CBR combination, we ran 10 iterations at 20 seconds simulator time each, and averaged the results.

Experiment 2

The purpose of experiment 2 was to test how fairly TCP variants interact with each other. In this experiment, we configured the network to have a CBR flow from Node 2 to Node 3, and TCP connections from Node 1 to Node 4 and from Node 5 to Node 6. In this portion, it was necessary for us to run the testing such as in the first experiment, but with two variants simultaneously. By doing so, we were able to judge how fairly the two variants ran together. For this experiment, we randomized the start times for each tcp variant between [0.0, 10.0) seconds, and used the random_ property for the CBR flow to randomize packet creation intervals. We each TCP set/CBR combination 10 times for 30 seconds in simulator time each. The TCP combinations used were “Reno/Reno”, “Reno/NewReno”, “Vegas/Vegas”, and “Vegas/NewReno”. CBR flow was incremented from 0 to 10.5 Mbps in steps of 0.5 Mbps.

Experiment 3

The purpose of experiment 3 was to test how queuing influenced the TCP variants, so for each TCP type, we used changed the queuing algorithm used by each node to be either RED (Random Early Drop) or DropTail. Additionally, rather than varying the CBR flow, we recorded flow statistics on a 250 ms interval. We focused on observing how TCP Reno and SACK flows behaved under the two queuing algorithms. For this experiment, we randomized the start time of the CBR flow, and used the random_ property to randomize packet creation for the CBR flow and the TCP flow. Since there were fewer combinations for this experiment, we were able to run 20 iterations, each running for 10 seconds in simulator time.

Results

In the following section, we describe the results obtained from the experiments.

Experiment 1

In the first experiment, we observed how increasing the CBR rate affected the performance of the TCP variants. In order to test how the variants reacted under congestion, throughout this experiment, we slowly increased the bandwidth of the CBR flow until it reached a bottleneck capacity. We observed the throughput, packet drop rate, and latency throughout this process.

In the results for the throughput as the CBR flow increased, the general pattern of all the TCP variants look similar. However, according to the graph, it is apparent that the TCP variants Tahoe, Reno, and NewReno

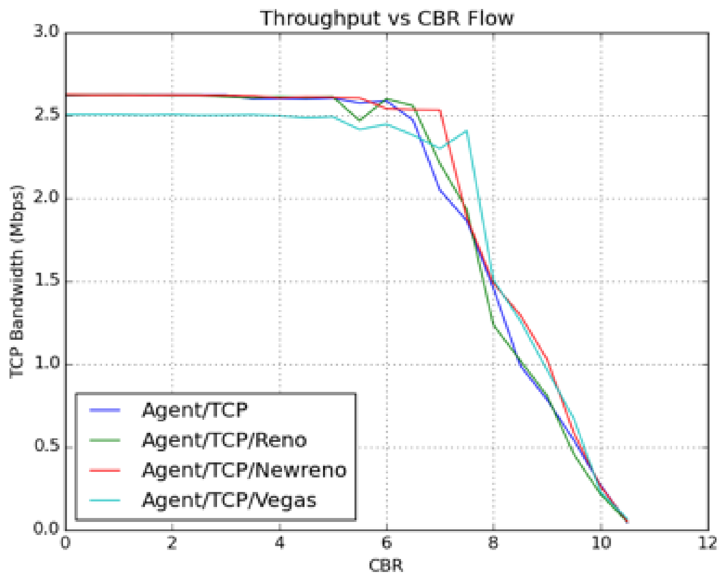


Figure 2

seemed to have slightly higher bandwidths than TCP Vegas when the network is not under contention.

Such as with the bandwidth results, the RTT and packet drop results seemed to stay consistent among all the variants. However, it was noticeable that as the CBR increased, the RTT of NewReno was longest at around 500 ms, while the RTT of Vegas reached it's limit significantly lower at around 300 ms.

Experiment 2

In the second experiment, we wanted to analyze the fairness of the variants when competing in a network. In order to implement this, we ran the test using a combination of two TCP variants for each trial.

This was done by testing the following combinations of variants:

- Reno/Reno
- NewReno/Reno
- Vegas/Vegas
- NewReno/Vegas

Generally, our results for Experiment 2 showed that the TCP variants examined act fairly.

The Reno/Reno results showed consistency and a very small level of variance.

The testing between the combination of two Vegas runs also resulted in similar results as the Reno/Reno run. The results generally showed fairness and the two TCP variants showed consistencies.

This was similar in the NewReno/Reno testing. However, while the drop rate and RTT showed consistency between Reno and NewReno, there was a distinction between the behaviors in the throughput between Reno and NewReno.

As shown in Figure 3, as CBR increases, there seems to be a steeper decline in Reno than in NewReno. This is likely due to the improvement made in NewReno, which requires the protocol to stay in Fast-recovery until all previously unacknowledged packets are acked--thus preventing multiple cwnd reductions in a row.

While there was not much distinction between the previous tests, this was not the case with NewReno and Vegas.

First, NewReno showed a slightly better throughput under low CBR flow with 2.5 Mbps vs 2.7 Mbps.

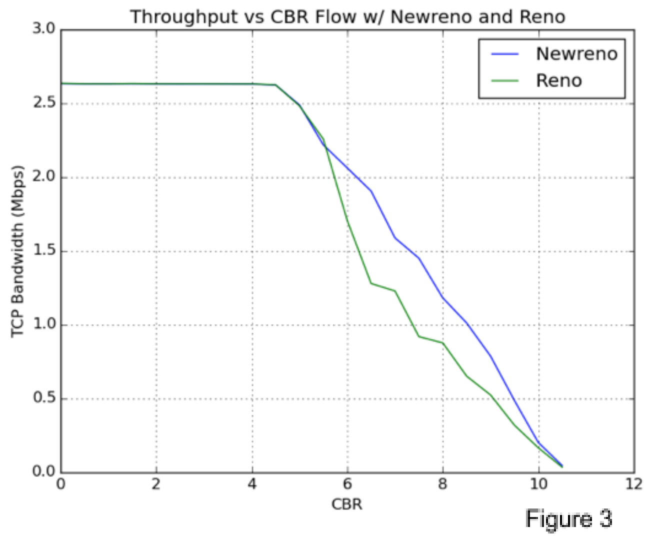


Figure 3

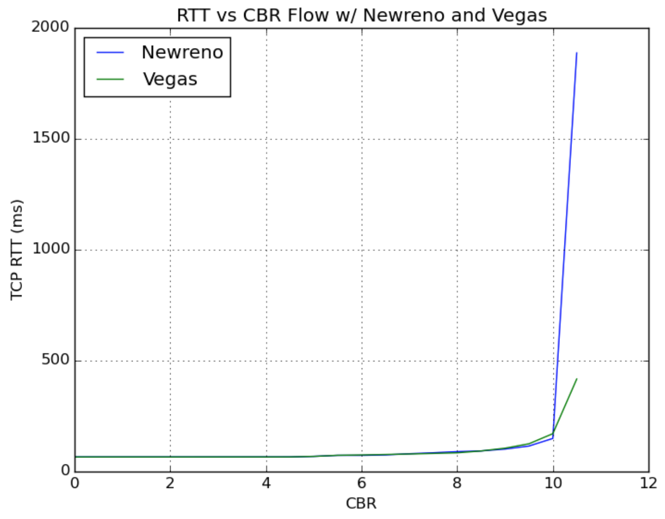


Figure 4

NewReno also maintained this edge under higher CBR flow, with a slower, steadier reduction in throughput at a rate of about 1 Mbps lost per 1 Mbps added to CBR flow. On the other hand, as the CBR flow increased, the rate at which NewReno dropped packets increased more rapidly than Vegas. This difference in fairness can be seen in the comparison of the results between the RTT vs CBR flow. As shown in fig. 4 it can be seen that as CBR reaches bottleneck capacity NewReno's RTT increases rapidly while there is a more steady increase from Vegas. Note that under most network situations, the two have near identical RTT.

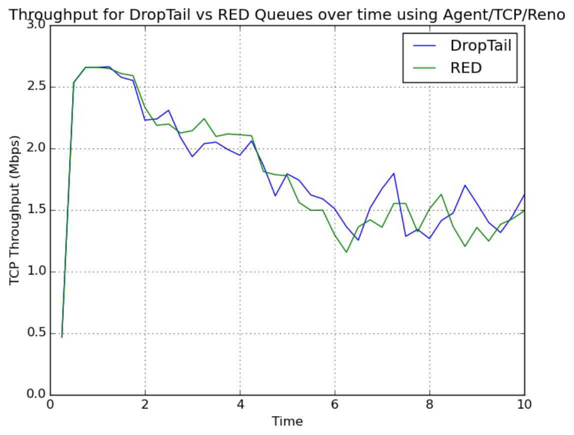
Figure 5

Experiment 3

In the last experiment, we focused on the performance of TCP Reno and Sack under the influence of the RED (Random Early Drop) and DropTail queuing algorithms.

Overall, we found that RED and DropTail performed similarly to each other. In the initial stages of the test, both TCPs under both queue types rapidly reached their resting throughput at about 2.7 Mbps. This is expected, since the queues are unlikely to drop packets when there is no contention in the network. As the CBR activate around the 2 second mark, throughput begins to drop off. Flows under both queues decrease at approximately the same rate, however, it should be noted that throughput under RED is less erratic. Additionally, Sack performs better with RED than with DropTail. In the first few seconds of CBR traffic, Sack is less responsive under RED than under DropTail.

It also appears that RED has a higher drop-rate than DropTail, maintaining both a higher baseline at about 1 % and also having spikes with a much higher magnitude than DropTail. RTT did not differ between the queues.



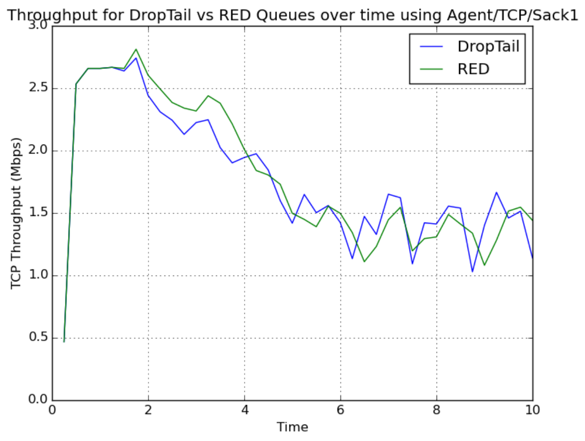


Figure 6, 7

Conclusions

In this paper, we used NS-2 to perform different simulations using the TCP variants of TCP, TCP Reno, TCP NewReno, and TCP Vegas. Generally, the results did not show many significant differences except in certain scenarios. Experiment 1 showed fairly consistent results between all the different variants. Experiment 2 showed that most TCP variants showed fairness when running simultaneously, with the exception of TCP Vegas. This could be due to the fact that TCP Vegas is built in with congestion avoidance.

In the occasion when the network is becoming more congested, Vegas may behave to avoid this congestion.

References

CS4700 Spring 2015 Lecture Notes