Performance Comparison of Various TCP Versions

Weihan Liu U93974332, Jingxin Dai U69300525

GitHub link: https://github.com/MrWickham/CS655_GENI_Mini_Project_TCP_Version

Project on GENI and any public link: **GENI slice name:** project-tcpVersion

Slice link: https://portal.geni.net/secure/slice.php?slice_id=d0c8e739-022c-4d3b-90c0-8898354250f5

1. Introduction / Problem Statement

In GENI TCP lab, we have learned and experimented with how to change the TCP flavor to Reno and measure throughput using `iperf` for different values of packet loss and delay using `ping`.

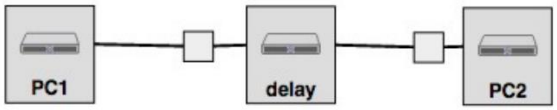
With the GENI Mini Project, we have extended this lab to measuring, comparing and analysis on metrics like delay, throughput, fairness and more, of different flavors of TCP, e.g. Reno, Cubic, Vegas, and BBR.

This project could help us develop a better understanding of behaviors, pros and cons of each flavor, and thus have a better idea of how different flavors would apply to different use cases better.

2. Experimental Methodology

Part 1: Throughput and delay

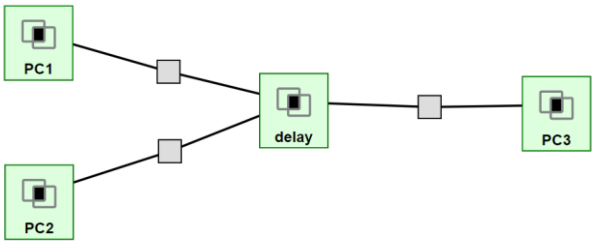
Based on the GENI TCP lab, we reserved resources as the topology as shown below:



The TCP versions of delay and PC2 would remain default as `cubic`. The methodology is to change TCP version of PC1 to Reno, Cubic and Vegas, then measure its throughput and delay to PC2. In addition, we could also measure those to the host delay to see if there are any new findings.

Part 2: Fairness

Part 2 would require adding one PC3 as node connected to delay and let both PC1 and PC2 sending to PC3 at the same time, as the topology below, which is currently reserved on slice.



Both the TCP versions of delay and PC2 would remain default as `cubic`. Each test would be based on the combination of TCP version of PC1, as well as PC2, being one of Reno, Cubic, Vegas and BBR. PC1 transmits data to PC3, after an interval of around 5 seconds, PC2 starts to send data to PC3. The first client to start transmission utilizes the network without restraint and is only required to share bandwidth when an additional client starts transmission.

The results would be presented as a 4×4 matrix of average throughputs of the TCP variants, along with some graphs of throughputs changing over time that worth noticing and analysis.

All of the metrics require at least five experiments of the same flavor to produce a reliable result. Statistics and diagram would be helpful to understand differences among different flavors and drawing conclusions.

3. Results

3.1 Usage Instructions

Changing TCP variants: After SSH, change the TCP flavors in PC1 and PC2 using command:

```
sudo sysctl net.ipv4.tcp_congestion_control=reno
```

This would change the TCP version to Reno.

Measuring throughput: use PC2 as server and run `iperf -s`, then use PC1 as client and run `iperf -c pc2 -t 50` to measure the throughput value.

Measuring delay: use `ping` command at PC1: `ping pc2 -c 10` to measure RTT.

3.2 Analysis

Part 1:

Figure 1: Delay and throughput comparison of TCP variants

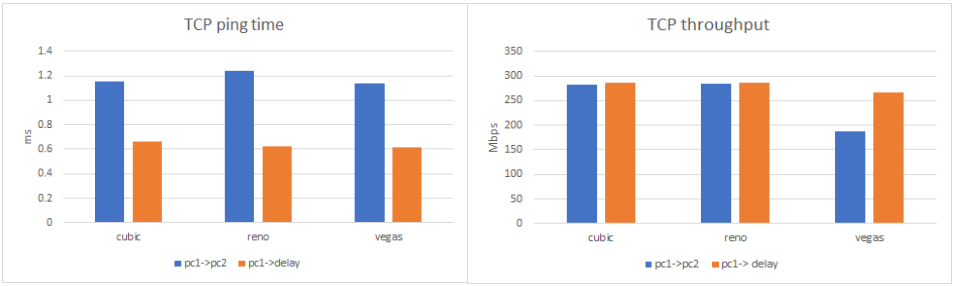


Figure 1 shows delay and throughput comparison of TCP variants from PC1 to PC2 and to delay. For delay, we can see the results are very similar. While for throughput, we can see that from PC1 to PC2, Cubic and Reno are able to utilize almost all of available throughput, achieving good performance, while Vegas's throughput is only around two thirds of the other two.

However, while examining the throughput from PC1 to delay, we found out that the gap between throughput of Vegas and that of Cubic and Reno has been narrowed. This gave us the conjecture that throughput of Vegas will degrade if there are more hops between source and destination.

Part 2:

Table 1: Throughput comparison of TCP variants

	Cubic	Reno	Vegas	BBR
Cubic	49.5778	47.2111	92.4111	50.95778
Reno	47.5333	49.48889	92.6	53.33625
Vegas	2.52	2.624	48.7333	25.80078
BBR	45.80333	32.48889	71.7	49.4333

Table 1 presents the average throughputs of the TCP variants. Thus when a Cubic connection is opened followed by an Vegas connection, Cubic attains an average throughput of 92.4111 Mbps and Vegas 2.52 Mbps.

We can see that all four variants can reach fairness with itself. Cubic and Reno seem to be sharing throughput evenly with each other. If ran along with Cubic or Reno, the ratio of throughput with Vegas would be 92:2, and with BBR 71:25. It would be fair to say that it's difficult to keep the fairness for Vegas if it's not sharing with other variants.

Figure 3: Throughput of TCP variants changing over time

Another interesting discovery from the figures above recording change of throughput over time is that, while the combination of Cubic, Reno and Vegas tend to become stable relatively quickly, if any hosts is using TCP BBR, the throughputs of two hosts would change dynamically. We know from Table 1 that the overall throughput of BBR is slightly above that of Cubic and Reno, and leaves Vegas a relatively generous space of 25.8 Mbps rather than 2.5 Mbps that Cubic and Reno do.

4. Conclusion

Comparing the congestion control algorithms to each other shows that whilst some of the algorithms can co-exist, others cannot. TCP-Vegas consistently had a low mean throughput of about 10 Mbps against all other TCP variants. TCP-Vegas is perhaps the algorithm that is most sensitive to network congestion, as it gives up bandwidth the most easily. CUBIC, and Reno share the available bandwidth more or less equally among themselves. BBR dynamically adjusts the throughput to make sure that overall fairness is achieved.

This project could be further extended to including new protocols like Versus, Sprout to further comparison.

5. Division of Labor

Weihan Liu: Experiment and report for part 2; GitHub and video instruction

Jingxin Dai: Experiment and report for part 1; Tables and diagrams