Performance Comparison of various TCP versions

Yifan Guo(guoyf1@bu.edu, BU ID:U68321686)
Feifan Yuan(ffyuan@bu.edu, BU ID:U61632796)
Ziyang Sheng(shengzy@bu.edu, BU ID: U95910439)

December 9, 2022

Github Link: https://github.com/yuanfeif/CS655_GENI_Project
Demo Video Link: https://drive.google.com/drive/folders/1WxSuiYberBQ8F1ET8bEQboAsFZnvm3t4?usp=sharing

1 Introduction

Since TCP carries so much traffic, different congestion control algorithms have been developed to prevent the Internet from being stuck in low-efficiency due to over-utilization. The basic idea of TCP congestion control algorithms is additive increase multiplicative decrease(AIMD), in which the sender decreases the sending rate when the network is over-utilized, and increases the sending rate when the network is under-utilized.

Among those congestion control algorithms, TCP Reno was first proposed to avoid congestion situations by reducing its congestion window by a multiplicative factor. However, TCP Reno suffered from a very critical backlog. It may reduce the congestion window multiple times for multiple packet loss in the same window, which hurts its performance. Therefore, people kept optimizing the algorithm and derived multiple new versions, such as TCP Cubic, the current default on Linux servers; TCP Vegas, the delay-based congestion control; and TCP BBR, the most recent one proposed by Google.

This project is to analyze four different versions of TCP(Reno, Cubic, Vegas, BBR) by measuring and comparing some metrics, such as throughput, delay and fairness, under the same network environment, so that we can have a better understanding of different congestion control algorithms and their advantages and disadvantages.

2 Experimental Methodology

2.1 Throughput and Delay

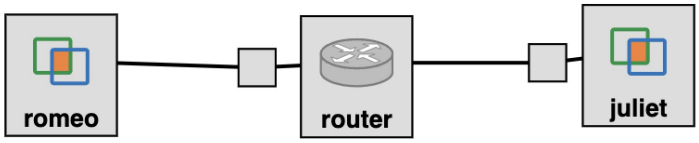

Figure 1: Resources view for 2.1

To measure throughput and delay, we reserved a topology including one sender host(romeo), one receiver host(juliet), and a router between them. The router will buffer traffic between the sender and the receiver. If the buffer in the router becomes full, it will drop packets, triggering TCP congestion control behavior. The TCP version of the router and the receiver remains default as cubic. We then change TCP version of the sender in between Reno, Cubic, Vegas, and Bbr, then record its throughput and delay both to the router and to the receiver. In addition, we also configured the router as a 1Mbp bottleneck, along with a buffer size of 0.1MB, in both direction. findings.

2.2 Fairness

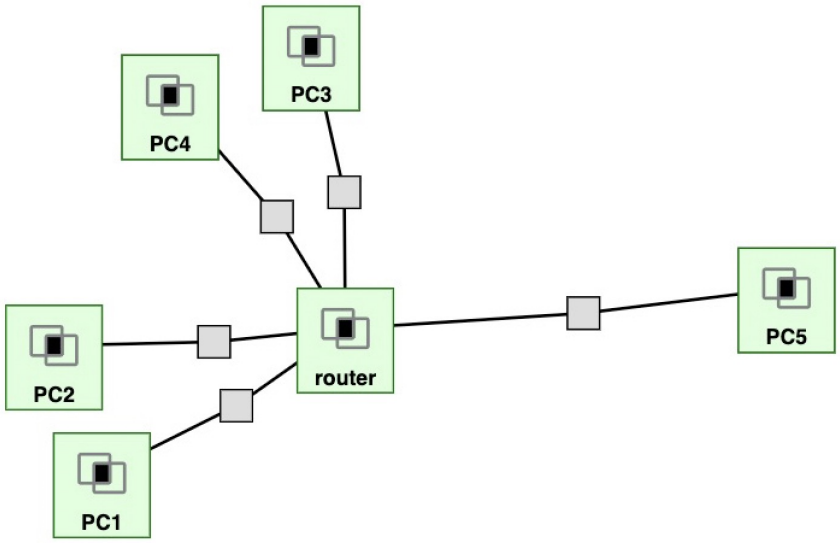

Figure 2: Resources view for 2.2

To measure fairness, we reserved a topology including four sender hosts(PC1, PC2, PC3, PC4), one receiver host(PC5), and a router between them. The TCP version of the router and the receiver remains default as cubic. Each client is allocated with a unique kind of TCP version(PC1:Reno, PC2:Cubic, PC3:Vegas, PC4:Bbr). The first client to start transmission utilizes the network without restraint and is only required to share bandwidth when an additional client starts transmission. Therefore, we start the transmission of four clients simultaneously to record the throughput for each of them.

3 Results

3.1 Usage Instructions

The default settings of the test includes the following:

- Operating System: Mac OS
- Network Environment: GENI

3.1.1 preparation

For the preparation part, we first log into each end host and install the iperf network testing tool, with the command:

```
sudo apt-get update
sudo apt-get -y install iperf3
```

Also we need to configure an additional setting:

```
sudo sysctl -w net.ipv4.tcp_no_metrics_save=1
```

Also, configure the router as a 1 Mbps bottleneck, with a buffer size of 0.1 MB, in both directions:

```
sudo tc qdisc del dev eth1 root
sudo tc qdisc add dev eth1 root handle 1: htb default 3
sudo tc class add dev eth1 parent 1: classid 1:3 htb rate 1Mbit
sudo tc qdisc add dev eth1 parent 1:3 handle 3: bfifo limit 0.1MB
```

```
sudo tc qdisc del dev eth2 root
sudo tc qdisc add dev eth2 root handle 1: htb default 3
sudo tc class add dev eth2 parent 1: classid 1:3 htb rate 1Mbit
sudo tc qdisc add dev eth2 parent 1:3 handle 3: bfifo limit 0.1MB
```

3.1.2 Throughput and Delay

For measuring the throughput and delay, we first need set the tcp flavor for the client, using the command:

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

You can substitute reno with other tcp flavors. After setting the tcp flavor, we need run the server juliet with the command:

```
iperf -s -i 10
```

which means show the result every 10 seconds.
After that, we could start to run the client romeo to send the message with the command:

```
iperf -c juliet -t 60 -i 10
```

which means sending the message to juliet in 60 seconds and showing the result every 10 seconds.
When using ECN to measure the throughput and delay, steps are very similar except the configuration of the router:

```
sudo tc qdisc del dev eth1 root
sudo tc qdisc add dev eth1 root handle 1: htb default 3
sudo tc class add dev eth1 parent 1: classid 1:3 htb rate 1Mbit
sudo tc qdisc add dev eth1 parent 1:3 handle 3:  codel limit 100 target 10ms ecn

sudo tc qdisc del dev eth2 root
sudo tc qdisc add dev eth2 root handle 1: htb default 3
sudo tc class add dev eth2 parent 1: classid 1:3 htb rate 1Mbit
sudo tc qdisc add dev eth2 parent 1:3 handle 3:  codel limit 100 target 10ms ecn
```

3.1.3 Fairness

From PC1 to PC4, we could set the tcp flavor into reno, cubic, vegas and Bbr with the command:

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

After that, we are going to run the server to prepare for receiving messages:

```
iperf -s -i 10
```

And we are going to run the clients simultaneously to send messages to the server:

```
iperf -c PC5 -t 60 -i 10
```

3.2 Analysis

As figure 3 shows, the throughput between the sender and receiver is basically the same as the predefined bottleneck, the TCP BBR has the highest throughput with more than 1020Kbps, while Reno has the lowest throughput with less than 970Kbps. However, in figure 4, the throughput between the sender and the router for different TCP versions are pretty much the same, except that TCP Cubic has a lower throughput than the other three.

As figure 5 and 6 show, the delay between the sender and receiver is around 0.9ms, and the TCP BBR has the lowest delay with less than 0.9ms, while TCP Vegas has the highest delay with almost 0.92ms. The delay between the sender and the router for different TCP versions vary from 0.48ms to 0.52ms, among which TCP Vegas has the highest delay.

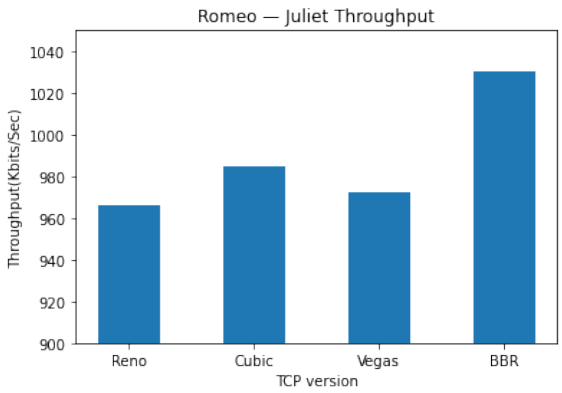

Figure 3:

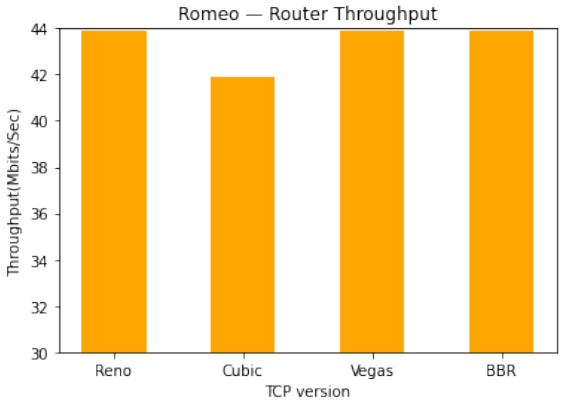

Figure 4:

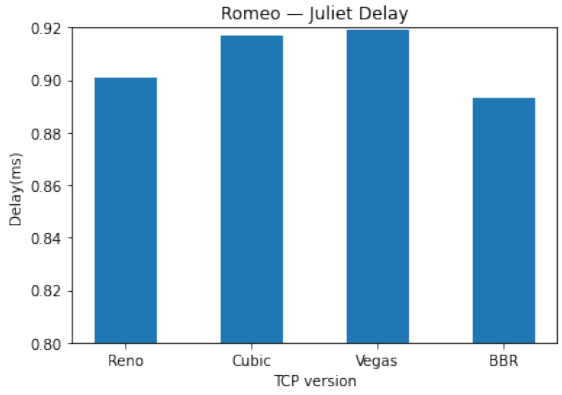

Figure 5:

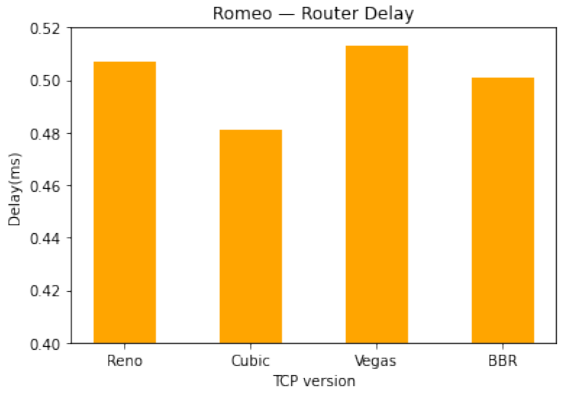

Figure 6:

As figure 7 shows, the throughput between the sender and receiver goes down when Explicit Congestion Notification(ECN) is applied, except that ECN increases the throughput of TCP BBR. Besides, in figure 8, ECN significantly lower the delay between the sender and the receiver for TCP Cubic, but it does not have much effect on the other three versions of TCP.

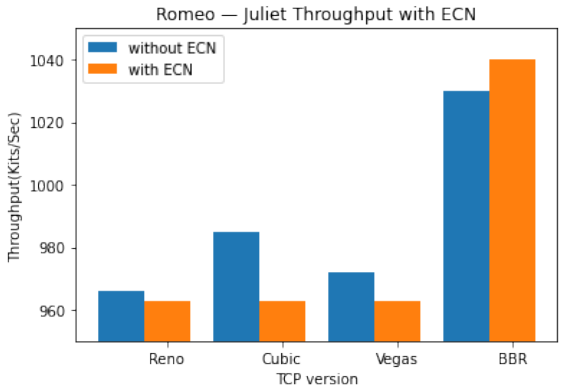

Figure 7:

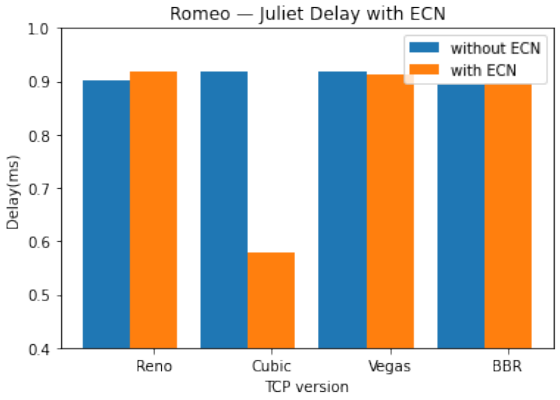

Figure 8:

For fairness, as figure 9 shows, the trends of throughput between the sender and receiver are quite similar. They all go down drastically at the beginning, and remain relatively stable for the rest of the time. Except that in TCP BBR, the throughput stays at a higher level than the other three, and it fluctuates all the time. In figure 10, the throughput percentage for different TCP versions are shown in the pie chart, where TCP BBR takes more than 60% while TCP Cubic has the lowest throughput, which only takes less than 10% in total.

Figure 9: Throughput for fairness

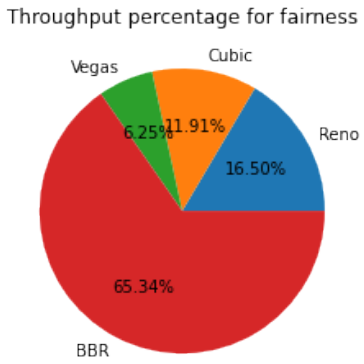

Figure 10:

4 Conclusion

Generally, TCP Reno and TCP Cubic are loss-based congestion control with the goal of high throughput, they tend to cause high queuing delays in the network, because they reduce their CWND only when they experience a packet loss. Compared to Reno, Cubic implements a binary search increase which can reach the available bandwidth much faster than Reno. And thus Cubic has a higher throughput in Figure 3.

Moreover, TCP Vegas is a delay-based algorithm that uses delay as a signal of congestion, and reduces its sending rate when the delay increases. TCP Vegas tends to limit throughput with very small RTT values, and overruns the connection in large RTT values. As a result, the throughput of TCP Vegas in Figure 3 is relatively low, and the delay in Figure 5 is higher than others.

Besides, TCP BBR tries to maximize throughput and at the same time minimize queuing delay in the network. BBR is model-based, using the maximum bandwidth and round-trip time to build a model of the capability of the network based on BDP (bandwidth-delay product). The BBR model is rate-based and delay-based. It is vaguely similar to TCP Vegas in its partial reliance on latency to detect congestion, without the downsides. Therefore, we can see that BBR has the best performance both on throughput and delay. And it also takes the most percentage of throughput in the fairness experiment. All the results have proven that the BBR should be the most efficient of the four versions.

Lastly, explicit congestion notification(ECN) is a feature that allows routers to explicitly signal to a TCP sender when there is congestion. This allows the sender to reduce its congestion window before the router is forced to drop packets, reducing retransmissions. It can also help the router maintain a minimal queue, which reduces queuing delay, which explains the delay reduction in Figure 8.

5 Work division

Yifan Guo: Experiment for section2.2, demo video
Feifan Yuan: Experiment for section2.1, demo video
Ziyang Sheng: Diagrams, Report