CS339 Lab5 Test `TCP` Performance

Student Name: Zhou Haoquan

Student ID: 519370910059

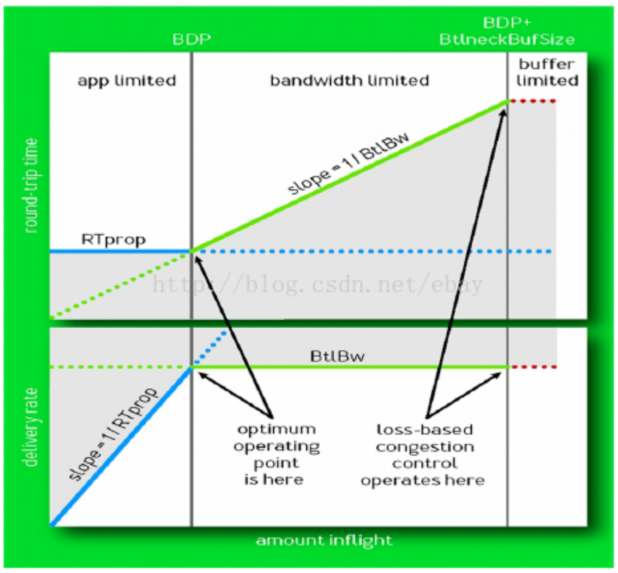
1. Pick one `TCP` congestion control algorithm other than `TCP Reno`, and explain how it works.

Here we pick and explain the `TCP BBR` congestion control algorithm. We start from the shortcoming of the `TCP Reno`:

- The traditional `TCP` protocol could not tell the difference between the duplicated `ACK` caused by packet loss and duplicated `ACK` caused by link error. As long as three duplicated `ACK`s are received, the sender will assume that the congestion happens in the network and reduce the `cwnd` to half of the original value. But in modern networks, especially a network adopts wireless connection, the rate of link error is so significant to be ignored. Always reduce the `cwnd` may no longer be a good idea, since it may waste some network resources.
- The traditional `TCP` protocol uses the packet loss as the signal of congestion. But this method tends to use up the buffers set in the network, which may cause a huge delay in the network.

Instead of pursuing the `loss-based congestion control`, the `TCP BBR` tries to find the `optimum operating point` without consuming the buffer in the network, just as shown in the following figure.

FIGURE 1: DELIVERY RATE AND ROUND-TRIP TIME VS. INFLIGHT



At the `optimum operating point` the number of packets in the network can be calculated as

$$BDP = maxBW \times minRTT$$

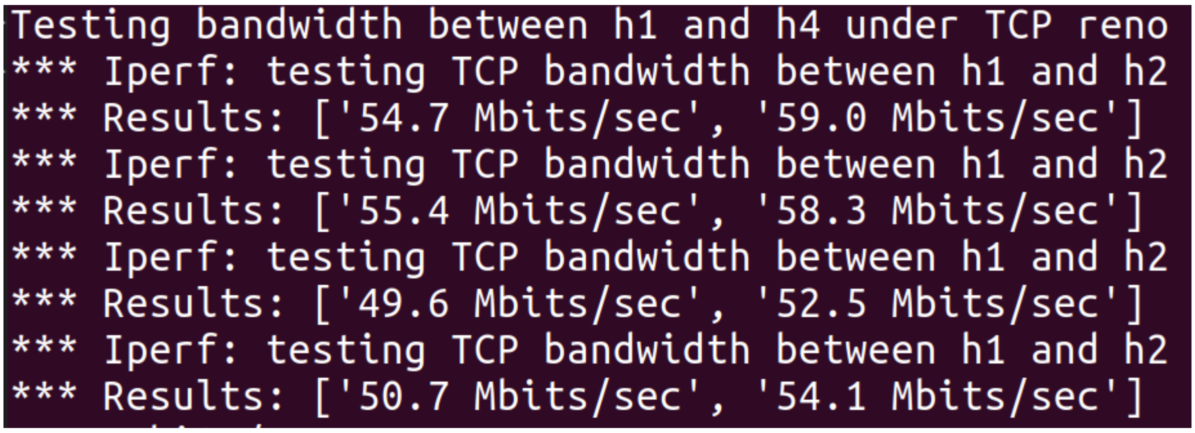
So in order to reach the `optimum operating point`, we should both test the `maxBW` and `minRTT`. However, if we want to measure the `maxBW`, we need to fill the bottleneck of the network, which may cause some packets stored in the buffer and cause the `RTT` to become large. And if we want to measure the `minRTT`, we may use as less packet as we want in the network, which may not use up the maximum bandwidth. In order to solve this problem, the `TCP BBR` choose to test the `maxBW` and `minRTT` separately in a sequential manner. That is, we may test the `maxBW` first and then `minRTT` in a short time. Then we treat them as if they are tested at the same time. The state of `TCP BBR` can be modeled in four steps:

1. **Slow start**. Just like traditional `TCP`, increase the `cwnd` in exponential form. While it finds that the transmission rate no longer grows for three times. It enter the **clear state**.
2. **Clear state**. Decrease the transmission rate in the same exponential form. Since we wait for three times in the last step, we will need to wait the buffer to clear two third of itself. Then we enter the next state.
3. **`Bandwidth` test**.
4. **`RTT` test**.

Note that the step 3 and 4 are done dynamically.

2. Enable `TCP Reno` and your selected TCP congestion control algorithm, and test them in `Mininet`.

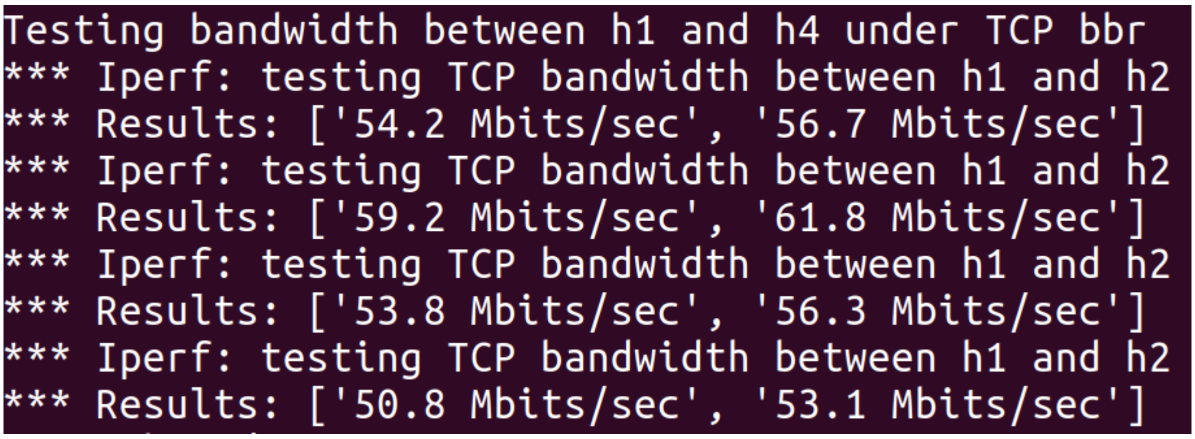
We first test the transmission rate for `TCP Reno`, the result is shown in the following figure.



```
Testing bandwidth between h1 and h4 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['54.7 Mbits/sec', '59.0 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['55.4 Mbits/sec', '58.3 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['49.6 Mbits/sec', '52.5 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['50.7 Mbits/sec', '54.1 Mbits/sec']
```

We tested it four times and we see that there is some fluctuation in the transmission rate. So we take the average value, which is roughly `54.29 Mbits/sec`.

We then test the transmission rate for `TCP BBR`, the result in shown in the following figure.



```
Testing bandwidth between h1 and h4 under TCP bbr
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['54.2 Mbits/sec', '56.7 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['59.2 Mbits/sec', '61.8 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['53.8 Mbits/sec', '56.3 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['50.8 Mbits/sec', '53.1 Mbits/sec']
```

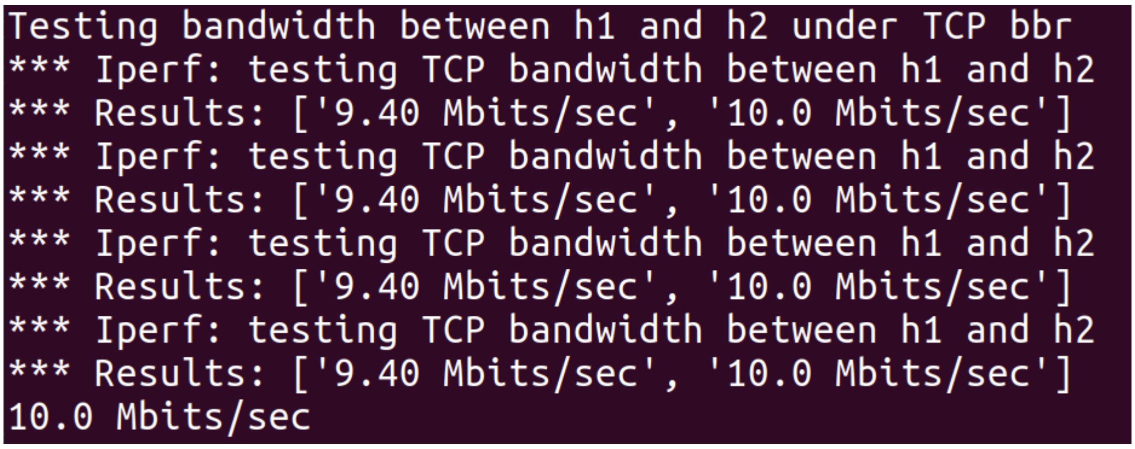
Again we do a average calculation and get the transmission rate to be `55.73 Mbits/sec`.

We see that these two congestion control algorithms are almost the same, with the `TCP BBR` to be slightly faster in this case.

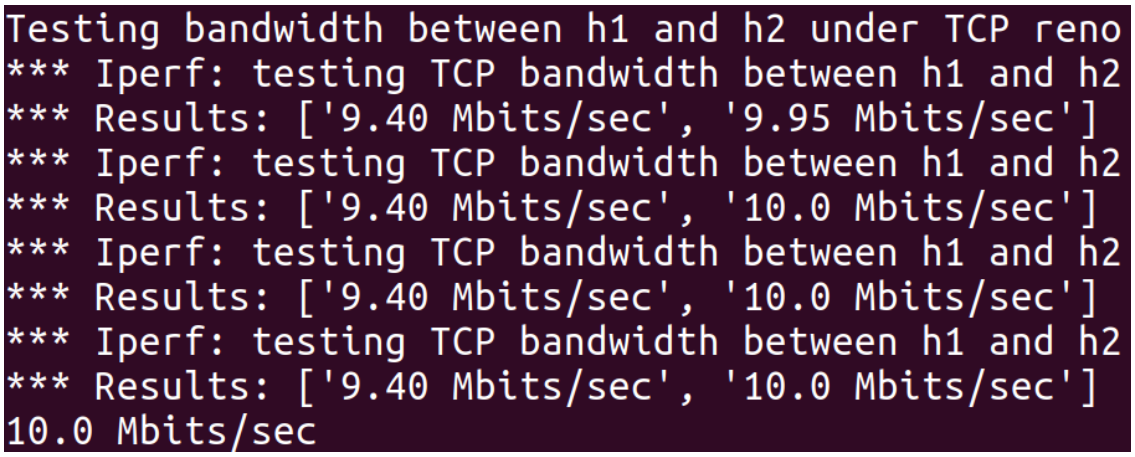
3. Construct a network with only one pair of sender and receiver. Study how TCP throughput varies with respect to link bandwidth/link delay/loss rate for the above two TCP versions.

3.1 Link Bandwidth

1. The `10Mbits/s` situation

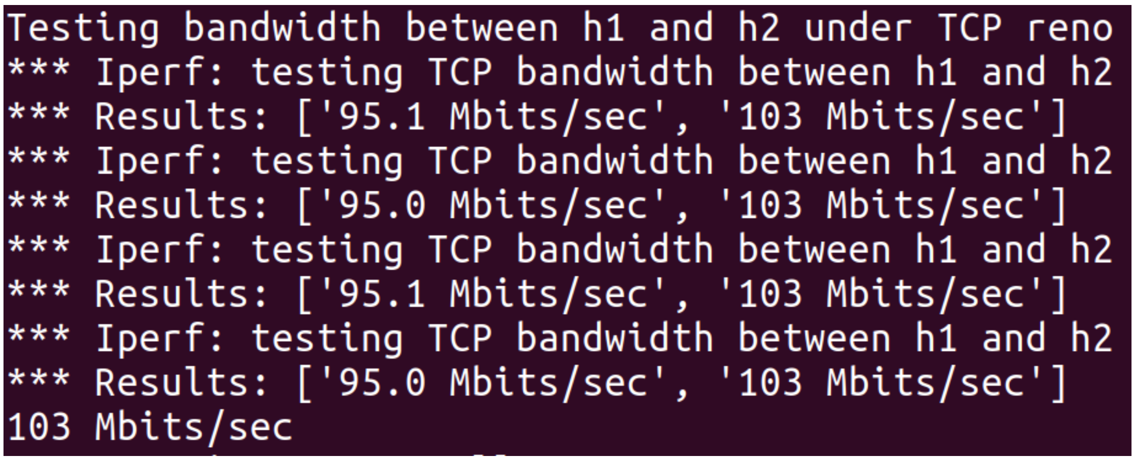


```
Testing bandwidth between h1 and h2 under TCP bbr
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.40 Mbits/sec', '10.0 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.40 Mbits/sec', '10.0 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.40 Mbits/sec', '10.0 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.40 Mbits/sec', '10.0 Mbits/sec']
10.0 Mbits/sec
```

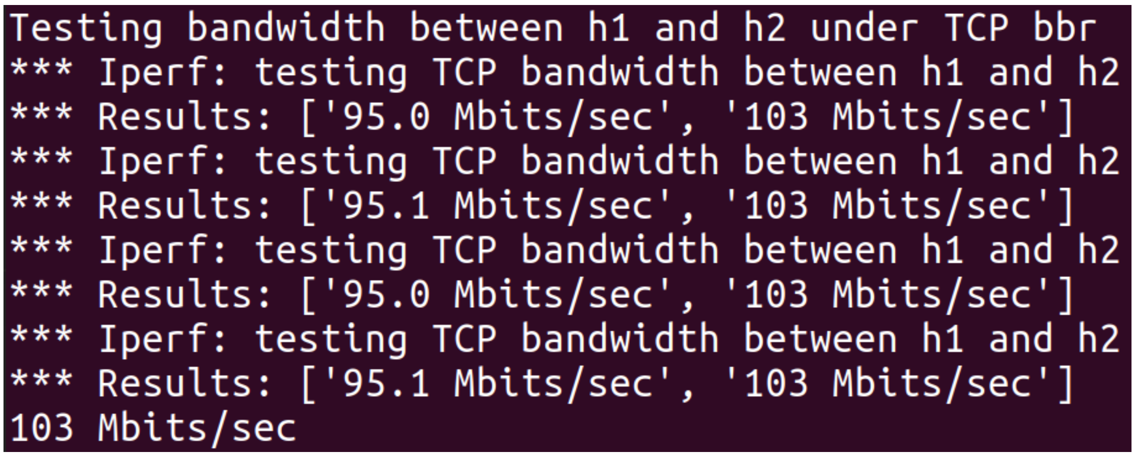


```
Testing bandwidth between h1 and h2 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.40 Mbits/sec', '9.95 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.40 Mbits/sec', '10.0 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.40 Mbits/sec', '10.0 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.40 Mbits/sec', '10.0 Mbits/sec']
10.0 Mbits/sec
```

2. The `100Mbits/s` situation



```
Testing bandwidth between h1 and h2 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['95.1 Mbits/sec', '103 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['95.0 Mbits/sec', '103 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['95.1 Mbits/sec', '103 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['95.0 Mbits/sec', '103 Mbits/sec']
103 Mbits/sec
```



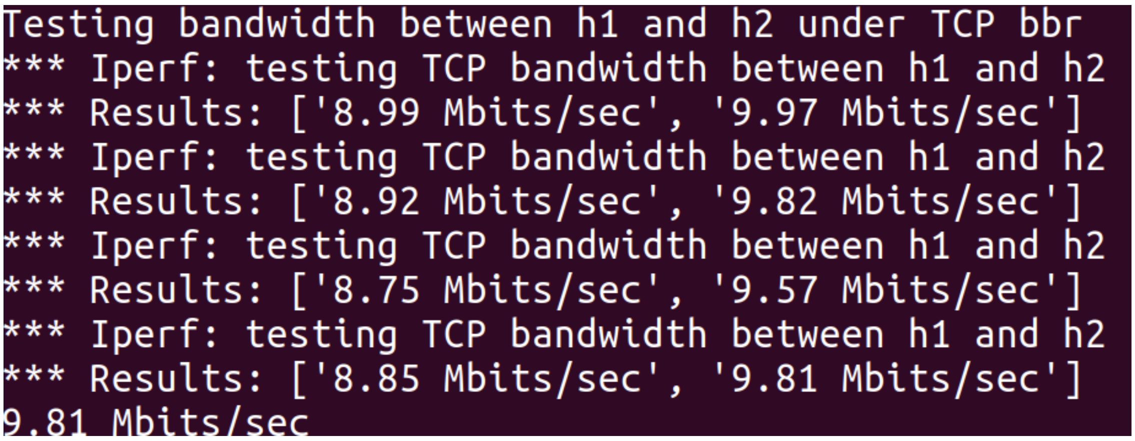
```
Testing bandwidth between h1 and h2 under TCP bbr
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['95.0 Mbits/sec', '103 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['95.1 Mbits/sec', '103 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['95.0 Mbits/sec', '103 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['95.1 Mbits/sec', '103 Mbits/sec']
103 Mbits/sec
```

we can see that a change of bandwidth does nearly no change to both congestion control algorithm.

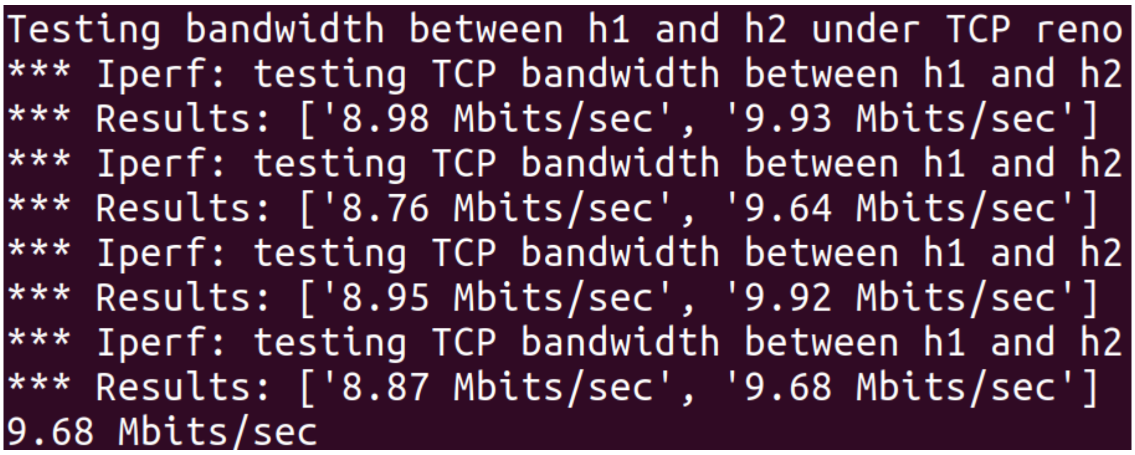
And the performance of them are almost the same.

3.2 Loss Rate

1. `5 %` case



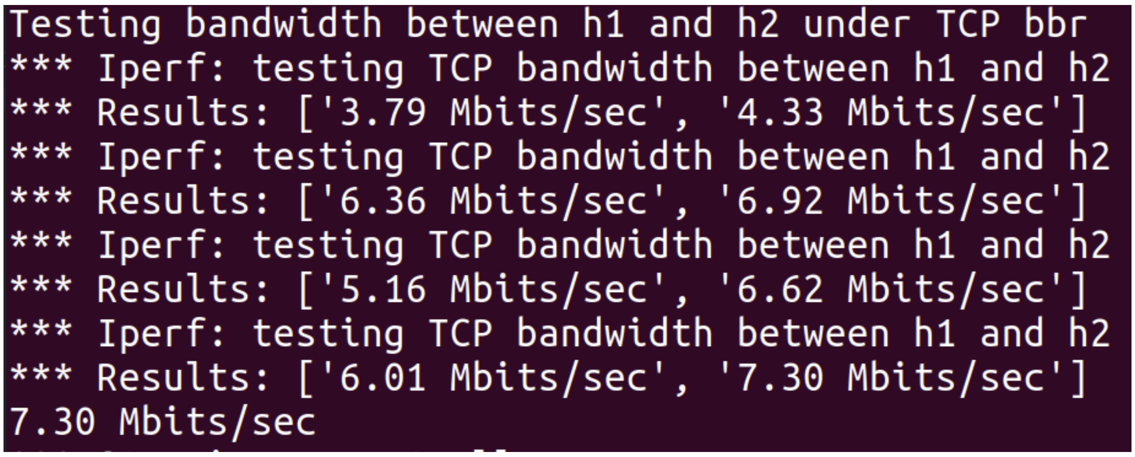
```
Testing bandwidth between h1 and h2 under TCP bbr
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.99 Mbits/sec', '9.97 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.92 Mbits/sec', '9.82 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.75 Mbits/sec', '9.57 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.85 Mbits/sec', '9.81 Mbits/sec']
9.81 Mbits/sec
```



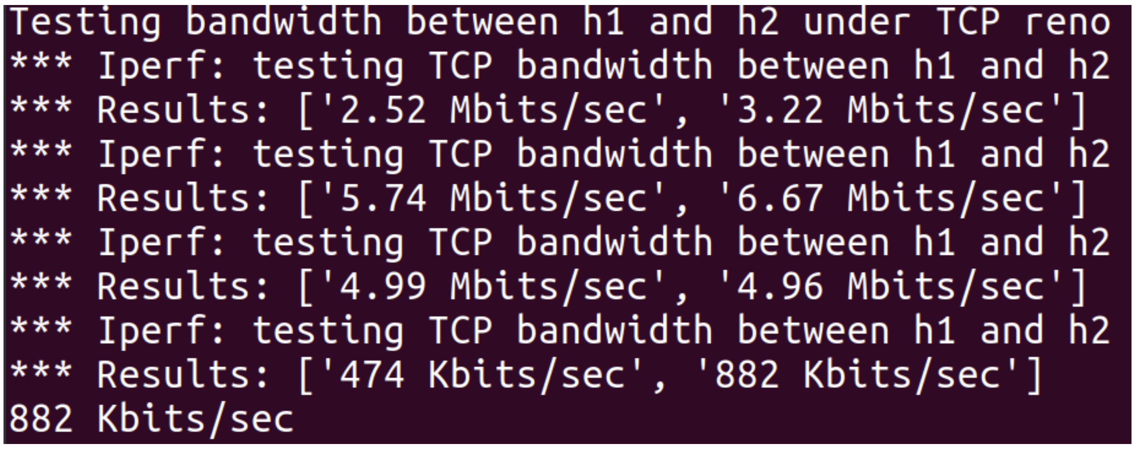
```
Testing bandwidth between h1 and h2 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.98 Mbits/sec', '9.93 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.76 Mbits/sec', '9.64 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.95 Mbits/sec', '9.92 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.87 Mbits/sec', '9.68 Mbits/sec']
9.68 Mbits/sec
```

In this `5%` loss rate testcase, both `TCP Reno` and `TCP BBR` see some shrinkage of bandwidth, but the difference between these two algorithms are not significant.

2. `10%` case



```
Testing bandwidth between h1 and h2 under TCP bbr
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['3.79 Mbits/sec', '4.33 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['6.36 Mbits/sec', '6.92 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['5.16 Mbits/sec', '6.62 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['6.01 Mbits/sec', '7.30 Mbits/sec']
7.30 Mbits/sec
```



```
Testing bandwidth between h1 and h2 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['2.52 Mbits/sec', '3.22 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['5.74 Mbits/sec', '6.67 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['4.99 Mbits/sec', '4.96 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['474 Kbits/sec', '882 Kbits/sec']
882 Kbits/sec
```

In this `10%` loss rate testcase, we see a significant difference between these two congestion control

algorithms. The average transmission rate during the stable time for the `TCP BBR` is about

`6.40Mbits/s`. However, the fluctuation for the `TCP Reno` algorithm is so large that the transmission rate at different time even differs for about 10 times. We focus on the relatively stable part and

calculate the average transmission rate to be `5.59 Mbits/s`.

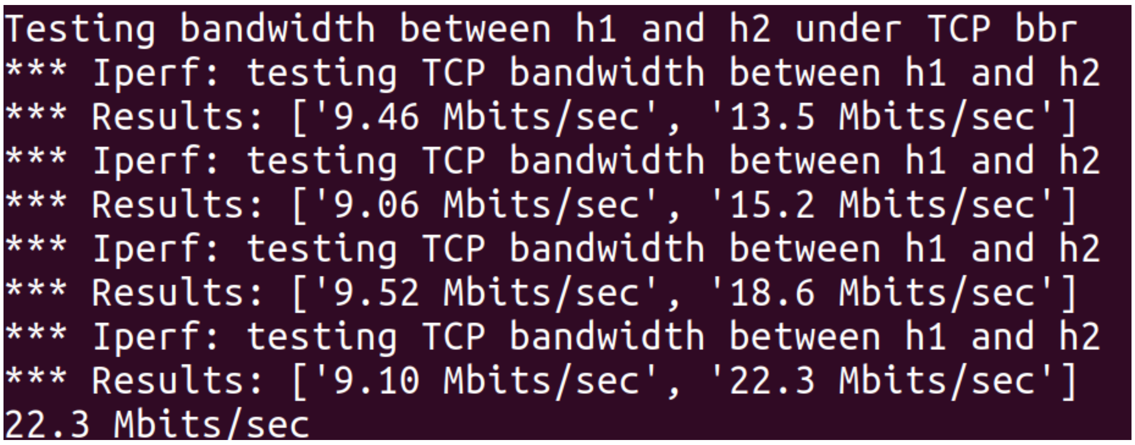
3. `20%` case

   In this case, the `TCP Reno` even fails to test the connection while the `TCP BBR` successfully tests the transmission rate in a long time.

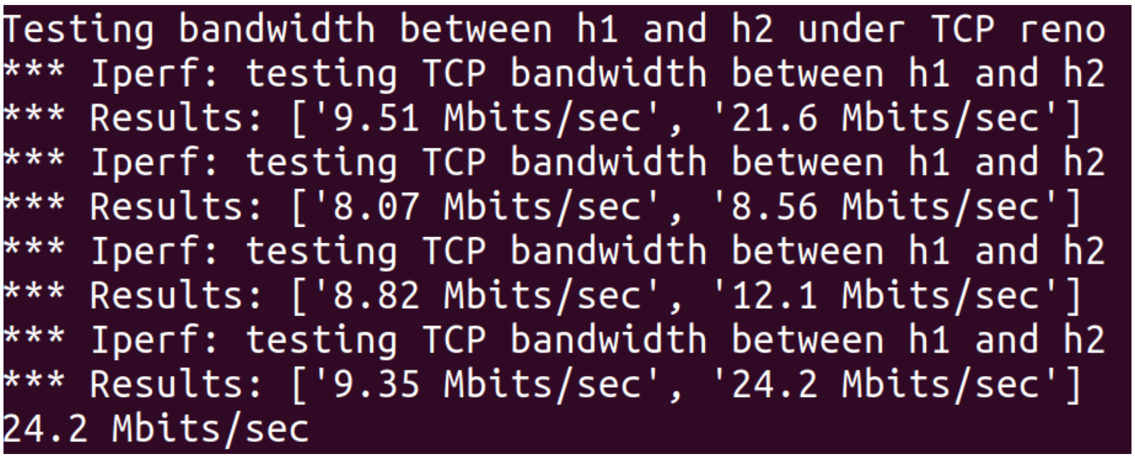
To conclude, the `TCP BBR` congestion control algorithm can manage the high loss rate better than the `TCP Reno` algorithm.

3.3 Link Delay

1. `10ms` delay case



```
Testing bandwidth between h1 and h2 under TCP bbr
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.46 Mbits/sec', '13.5 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.06 Mbits/sec', '15.2 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.52 Mbits/sec', '18.6 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.10 Mbits/sec', '22.3 Mbits/sec']
22.3 Mbits/sec
```

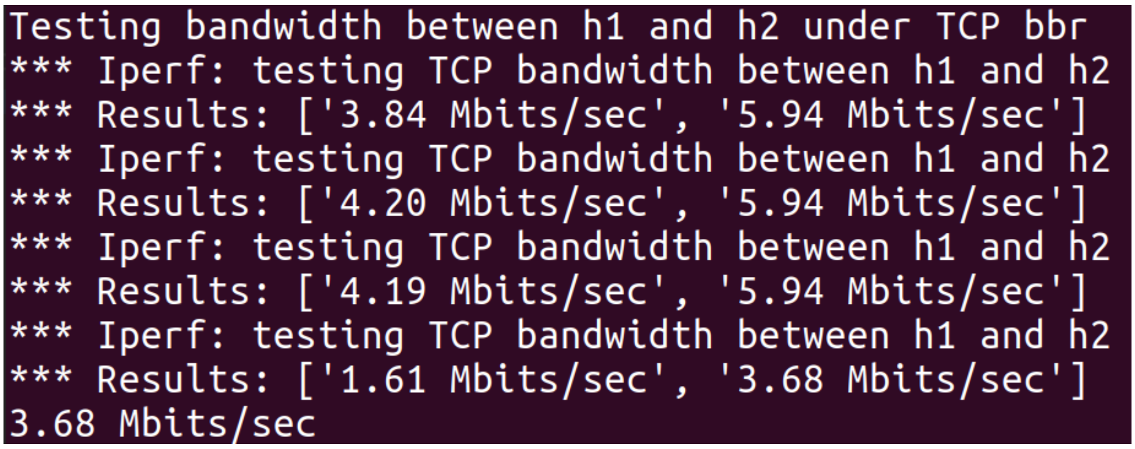


```
Testing bandwidth between h1 and h2 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.51 Mbits/sec', '21.6 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.07 Mbits/sec', '8.56 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['8.82 Mbits/sec', '12.1 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['9.35 Mbits/sec', '24.2 Mbits/sec']
24.2 Mbits/sec
```

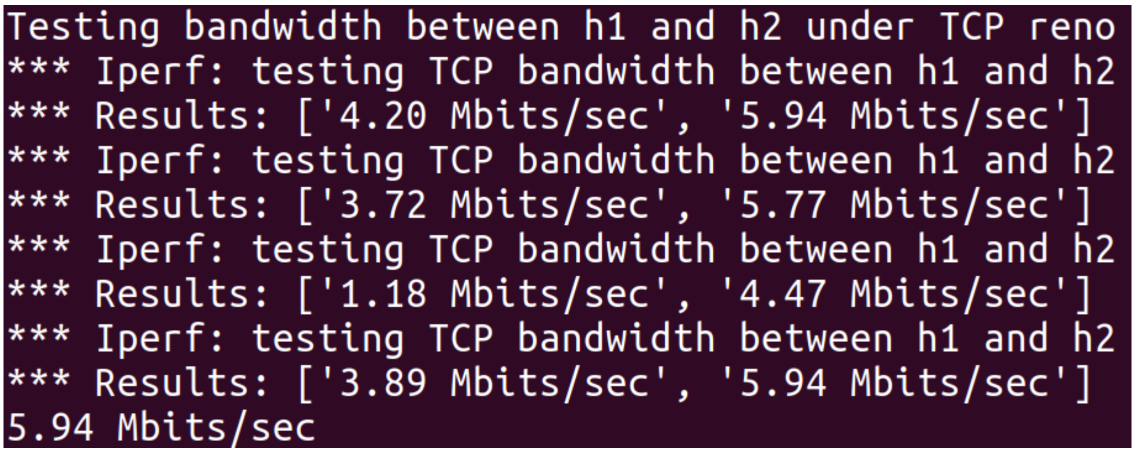
In this testcase, we can calculate the average transmission rate for `TCP BBR` to be `9.29 Mbits/s`.

While the average transmission rate for `TCP Reno` is `8.86 Mbits/s`. The difference between them is not large. The `TCP BBR` may have a slightly larger transmission rate.

2. `200ms` delay case



```
Testing bandwidth between h1 and h2 under TCP bbr
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['3.84 Mbits/sec', '5.94 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['4.20 Mbits/sec', '5.94 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['4.19 Mbits/sec', '5.94 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['1.61 Mbits/sec', '3.68 Mbits/sec']
3.68 Mbits/sec
```



```
Testing bandwidth between h1 and h2 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['4.20 Mbits/sec', '5.94 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['3.72 Mbits/sec', '5.77 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['1.18 Mbits/sec', '4.47 Mbits/sec']
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['3.89 Mbits/sec', '5.94 Mbits/sec']
5.94 Mbits/sec
```

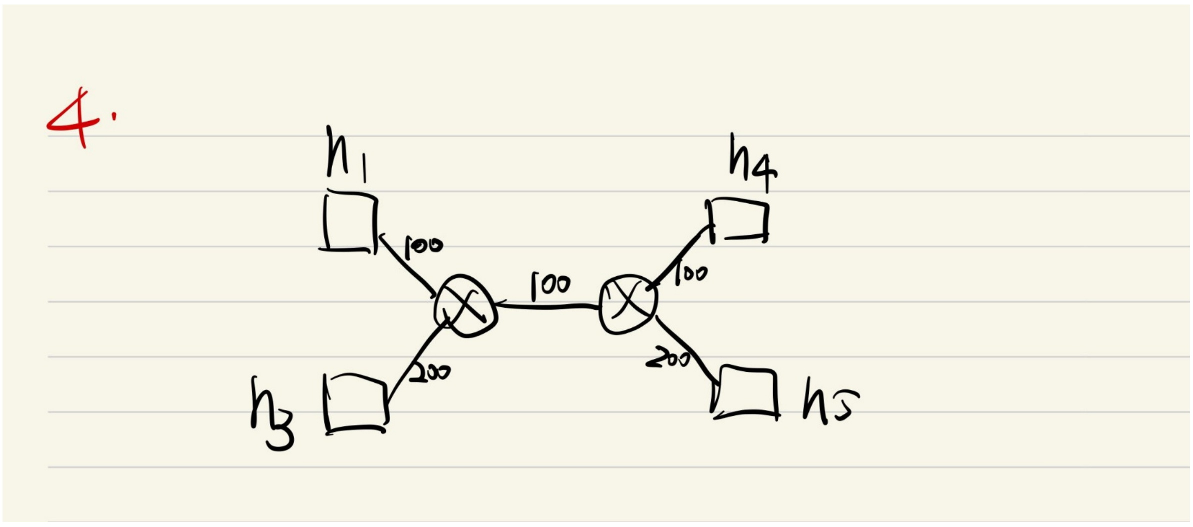
In this testcase, we can calculate the average transmission rate of `TCP BBR` to be `3.74 Mbits/s`.

While the average transmission rate of `TCP Reno` to be `3.77 Mbits/s`.

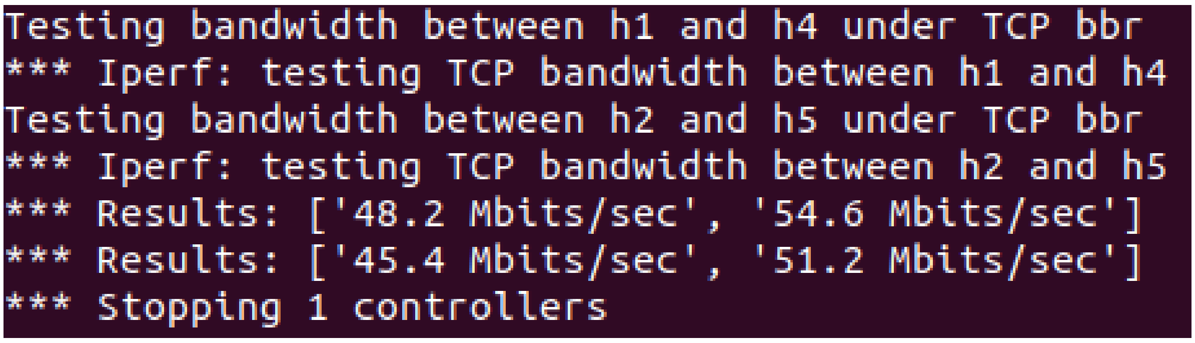
We see that there are still no significant difference between these two congestion control algorithms. We can conclude that the ability for these algorithms to handle the delay in network are almost the same.

5. Construct a network with a bottleneck link shared by multiple pairs of senders and receivers. Study how these sender-receiver pairs share the bottleneck link.

We construct a network with the topology as follows:



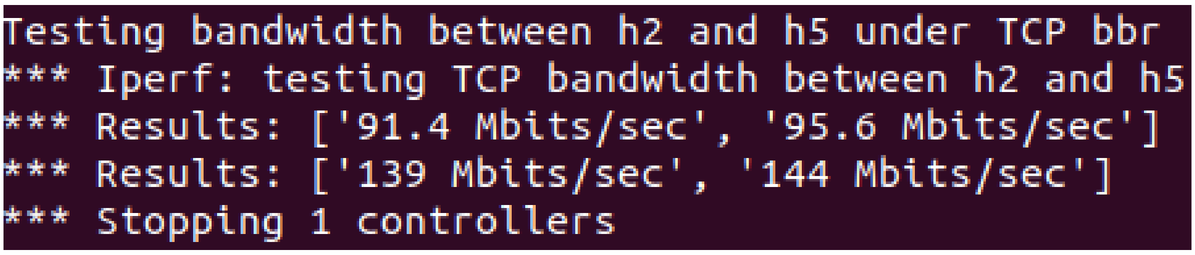
With `host1` sending message to `host4` and `host3` sending message to `host5`, we can conclude that the bottleneck here is the link between `switch3` and `switch4` with its bandwidth to be `200 Mbits/s`. Then we do the testing. The result is as follows:



```
Testing bandwidth between h1 and h4 under TCP bbr
*** Iperf: testing TCP bandwidth between h1 and h4
Testing bandwidth between h2 and h5 under TCP bbr
*** Iperf: testing TCP bandwidth between h2 and h5
*** Results: ['48.2 Mbits/sec', '54.6 Mbits/sec']
*** Results: ['45.4 Mbits/sec', '51.2 Mbits/sec']
*** Stopping 1 controllers
```

We see that even though this two hosts have different bandwidth, they share the same bandwidth when it comes to the bottleneck.

Now we try to increase the bandwidth of the bottleneck to be `250Mbits/s` and test the result again:



```
Testing bandwidth between h2 and h5 under TCP bbr
*** Iperf: testing TCP bandwidth between h2 and h5
*** Results: ['91.4 Mbits/sec', '95.6 Mbits/sec']
*** Results: ['139 Mbits/sec', '144 Mbits/sec']
*** Stopping 1 controllers
```

We see that the bandwidth is almost separated to `100 Mbits/s` and `150Mbits/s`, which fits the result of `Max-Min Fairness`. The `Max-Min Fairness`

algorithm will first increase both links at the same rate. However, when the link between `h1` and `h4` reaches maximum, it will continue to allocate the bandwidth to the other link, which makes the bandwidth between `h3` and `h5` to be `150Mbits/s`.