Lab5: Test TCP Performance

Xin Xu 519021910726

TCP Cubic

It is similar to Reno in stages Slow Start, Fast Retransmit and Fast Recovery. It is different from Reno in stage Congestion Avoidance. Rather than use a linear function to adjust congestion window size, cubic uses cubic function to adjust CongWin. The concrete steps of cubic are as follows:

Slow Start

When CongWin is below threshold, sender is in slow-start and CongWIn grows exponentially.

Congestion Avoidance

When there is a packet loss, the current CongWin will be marked as W_{max}. And the window is decreased by a factor $\beta = 0.2$ where β is a window decrease constant. Additionally, the regular fast recovery and retransmit of TCP are performed. After it enters into congestions avoidance where $CongWin > threshold$ from fast recovery, it starts to increase the window using the cubic function.

From that time a new CongWIn is calculated as:

$$W(t) = C(t-K)^3 + W_{max}$$

t is the time since the last window reduction. K is the time required by the window to reaches the value W_{max} with no loss, the equations are as follows:

$$K = \sqrt[3]{W_{max} * (\beta / C)}$$

$$C = 0.4 \text{ usually.}$$

Additionally, there is a standard TCP window size $W_{TCP}(t)$:

$$W_{TCP}(t) = W_{max} * (1-\beta) + 3 * \frac{\beta}{2-\beta} * \frac{t}{RTT}$$

The window grows as follow foe each ACK received:

- If $W(t) < W_{TCP}(t)$, then CUBIC TCP operates in support or friendly mode, so $CongWin = W_{TCP}(t)$.
- If $W(t) < W_{max}$, then it is in the concave region of the curve or stable mode. Then $CongWin = W(t)$.
- If $W(t) > W_{max}$, then it is in the convex region of the curve or exploratory mode. Then $CongWin = W(t)$.

Fast Retransmit

when the time-out expires or receiving 3 duplicates ACK, sender will retransmit the exact ACK packet.

Fast Recovery

When there is a packet loss(time out or 3 duplicate ACKs), the threshold will be set to $CongWin/2$. And the window is decreased by a factor $\beta = 0.2$.

Enable TCP Congestion Control Algorithm and Test in Mininet

according to the given steps, I have enabled different TCP algorithm in Ubuntu. The result is as follows:

```
littlestar@littlestar-VirtualBox:~$ sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = reno cubic vegas
littlestar@littlestar-VirtualBox:~$
```

As we can see, Reno, Cubic and Vegas are available in my Ubuntu.

Firstly, we test Reno, and the result is as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_2_reno.py
[sudo] password for littlestar:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 5ms delay 0.00000% loss) (h1, s1) (100.00Mbit 5ms de
lay 0.00000% loss) (100.00Mbit 5ms delay 0.00000% loss) (h2, s2) (100.00Mbit 200ms delay 0.10000% los
s) (100.00Mbit 200ms delay 0.10000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 200ms delay 0.10000% loss) (100.00Mbit 5ms
delay 0.00000% loss) (100.00Mbit 200ms delay 0.10000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['3.81 Mbits/sec', '5.76 Mbits/sec']
5.76 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

Then, we test Cubic, and the result is as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_2_cubic.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 5ms delay 0.00000% loss) (h1, s1) (100.00Mbit 5ms de
lay 0.00000% loss) (100.00Mbit 5ms delay 0.00000% loss) (h2, s2) (100.00Mbit 200ms delay 0.10000% los
s) (100.00Mbit 200ms delay 0.10000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 200ms delay 0.10000% loss) (100.00Mbit 5ms
delay 0.00000% loss) (100.00Mbit 200ms delay 0.10000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP cubic
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['36.3 Mbits/sec', '40.3 Mbits/sec']
40.3 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

How TCP Throughput Varies with One Pair of Sender and Receiver

Reno

At first, we set the bandwidth to maxmum and no delay, no loss as the comparative standard. The result is as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_reno_std.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h2, s2) (1000.00Mbit 0 delay 0.00000% loss) (100
0.00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['448 Mbits/sec', '449 Mbits/sec']
449 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

We can know that the bandwidth is 449 Mbits/sec.

Now, we reduce the bandwidth between s1 and s2 to 100 Mbits/sec, and the result is:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_reno_bw.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h2, s2) (100.00Mbit 0 delay 0.00000% loss) (100.
00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay
0.00000% loss) (100.00Mbit 0 delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h2 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['37.1 Mbits/sec', '37.2 Mbits/sec']
37.2 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

As we can see from the picture, the bandwidth is reduced to 37.2 Mbits/sec, which is decreased contrast to original 449 Mbits/sec.

To figure out the influence of delay, we set the link delay between h2 and s2 to 5 ms and the delay between s1 and s2 to 200ms. The result is as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_reno_delay.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 5ms del
ay 0.00000% loss) (1000.00Mbit 5ms delay 0.00000% loss) (h2, s2) (1000.00Mbit 200ms delay 0.00000% lo
ss) (1000.00Mbit 200ms delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 200ms delay 0.00000% loss) (1000.00Mbit 5ms
 delay 0.00000% loss) (1000.00Mbit 200ms delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['147 Mbits/sec', '148 Mbits/sec']
148 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

The bandwidth is reducesd to 148 Mbits/sec contrast to 449 Mbits/sec. Because the delay of bandwidth can lead to packets holding up, which limits the bandwidth.

To figure out the influence of loss rate, we set the loss rate between s1 and s2 to 0.1%. The result is:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_reno_loss.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h2, s2) (1000.00Mbit 0 delay 0.10000% loss) (100
0.00Mbit 0 delay 0.10000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.10000% loss) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.10000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['351 Mbits/sec', '351 Mbits/sec']
351 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

The bandwidth is reduced to 351 Mbits/sec. Because with packet loss, fewer data arrives in destination successfully, which lead to a low throughput and a low bandwidth.

Cubic

The process is the same with Reno.

At first, we construct a comparative standard of Cubic as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_cubic_std.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h2, s2) (1000.00Mbit 0 delay 0.00000% loss) (100
0.00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP cubic
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['445 Mbits/sec', '445 Mbits/sec']
445 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

We can know that the standard bandwidth of Cubic is 445 Mbits/sec, a little less than Reno.

Now, we reduce the bandwidth between s1 and s2 to 100 Mbits/sec, and the result is:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_cubic_bw.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h2, s2) (100.00Mbit 0 delay 0.00000% loss) (100.
00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay
0.00000% loss) (100.00Mbit 0 delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h2 under TCP cubic
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['63.0 Mbits/sec', '63.4 Mbits/sec']
63.4 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

As we can see from the picture, the bandwidth is reduced to 63.4 Mbits/sec, which is decreased contrast to original 445 Mbits/sec.

To figure out the influence of delay, we set the link delay between h2 and s2 to 5 ms and the link delay between s1 and s2 to 200ms. The result is as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_cubic_delay.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 5ms del
ay 0.00000% loss) (1000.00Mbit 5ms delay 0.00000% loss) (h2, s2) (1000.00Mbit 200ms delay 0.00000% lo
ss) (1000.00Mbit 200ms delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 200ms delay 0.00000% loss) (1000.00Mbit 5ms
 delay 0.00000% loss) (1000.00Mbit 200ms delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP cubic
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['145 Mbits/sec', '146 Mbits/sec']
146 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

The bandwidth is reducesd to 146 Mbits/sec contrast to 445 Mbits/sec. Because the delay of bandwidth can lead to packets holding up, which limits the bandwidth.

To figure out the influence of loss rate, we set the loss rate between s1 and s2 to 0.1%. The result is:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_cubic_loss.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h2, s2) (1000.00Mbit 0 delay 0.10000% loss) (100
0.00Mbit 0 delay 0.10000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.10000% loss) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.10000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP cubic
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['321 Mbits/sec', '321 Mbits/sec']
321 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

The bandwidth is reduced to 321 Mbits/sec. Because with packet loss, fewer data arrives in destination successfully, which lead to a low throughput and a low bandwidth.

Multiple Pairs Share the Bottleneck Link

I construct two pairs of sender and receiver to study the share of bottleneck link.

Reno

At first, I set the pair (h1, h2) is in the same condition with pair (h3, h4). The bandwidths of these two pairs are as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_4_reno_equal.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (h1, s1) (100.00Mbit 0 delay 0.
00000% loss) (100.00Mbit 0 delay 0.00000% loss) (h2, s2) (100.00Mbit 0 delay 0.00000% loss) (100.00Mb
it 0 delay 0.00000% loss) (h3, s1) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% lo
ss) (h4, s2) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us) h3 (cfs 150000/100000us) h4 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.
00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 dela
y 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s1-eth2
h4 h4-eth0:s2-eth2
Testing bandwidth between h1 and h4 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['79.1 Mbits/sec', '79.3 Mbits/sec']
*** Iperf: testing TCP bandwidth between h3 and h4
*** Results: ['64.3 Mbits/sec', '64.5 Mbits/sec']
The bandwidth between h1 and h2 is: 79.3 Mbits/sec
The bandwidth between h3 and h4 is: 64.5 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 5 links
.....
*** Stopping 2 switches
s1 s2
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

We can know that the bandwidth between h1 and h2 is 79.3 Mbits/sec, while the bandwidth between h3 and h4 is 64.5 Mbits/sec.

Now, we change the RTT of these two pairs by changing the link delay time. We increase the delay time between h1 and s1 to 5 ms and the delay time between h2 and s2 to 5 ms. It's regarded to reduce the bandwidth of (h1, h2). And the result is as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_4_reno_delay.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 5ms delay 0.00000% loss) (h1, s1) (100.00Mbit 5ms de
lay 0.00000% loss) (100.00Mbit 5ms delay 0.00000% loss) (h2, s2) (100.00Mbit 0 delay 0.00000% loss) (
100.00Mbit 0 delay 0.00000% loss) (h3, s1) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.0
0000% loss) (h4, s2) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us) h3 (cfs 150000/100000us) h4 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay
0.00000% loss) (100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0
delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s1-eth2
h4 h4-eth0:s2-eth2
Testing bandwidth between h1 and h4 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['36.2 Mbits/sec', '37.0 Mbits/sec']
*** Iperf: testing TCP bandwidth between h3 and h4
*** Results: ['69.0 Mbits/sec', '69.1 Mbits/sec']
The bandwidth between h1 and h2 is: 37.0 Mbits/sec
The bandwidth between h3 and h4 is: 69.1 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 5 links
.....
*** Stopping 2 switches
s1 s2
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

As we can see, the bandwidth between h1 and h2 is 37.0 Mbits/sec, while the bandwidth between h3 and h4 is 69.1 Mbits/sec. That's because RTT of (h1, h2) becomes larger than (h3, h4) owing to link delay. The speed to increase CongWin in (h1, h2) is much slower than (h3, h4), thus (h1, h2) is less competitive to preempt bandwidth only to holds less bandwidth. In this way, Reno is unfair.

Cubic

It is the same with Reno. In equal condition, the bandwidths of these (h1, h2) and (h3, h4) are as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_4_cubic_equal.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (h1, s1) (100.00Mbit 0 delay 0.
00000% loss) (100.00Mbit 0 delay 0.00000% loss) (h2, s2) (100.00Mbit 0 delay 0.00000% loss) (100.00Mb
it 0 delay 0.00000% loss) (h3, s1) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% lo
ss) (h4, s2) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us) h3 (cfs 150000/100000us) h4 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.
00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 dela
y 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s1-eth2
h4 h4-eth0:s2-eth2
Testing bandwidth between h1 and h4 under TCP cubic
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['78.3 Mbits/sec', '79.0 Mbits/sec']
*** Iperf: testing TCP bandwidth between h3 and h4
*** Results: ['82.1 Mbits/sec', '82.4 Mbits/sec']
The bandwidth between h1 and h2 is: 79.0 Mbits/sec
The bandwidth between h3 and h4 is: 82.4 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 5 links
.....
*** Stopping 2 switches
s1 s2
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

We can know that the bandwidth between h1 and h2 is 79.0 Mbits/sec, while the bandwidth between h3 and h4 is 82.4 Mbits/sec.

In this term, we also increase the RTT of (h3, h4) by setting the link delay time. We increase the delay time between h3 and s1 to 5 ms and the delay time between h4 and s2 to 5 ms. Because Cubic's CongWin update is independent of RTT, so we guess it will achieve a better distribution of bandwidth than Reno.

The result is as follows:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_4_cubic_delay.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (h1, s1) (100.00Mbit 0 delay 0.
00000% loss) (100.00Mbit 0 delay 0.00000% loss) (h2, s2) (100.00Mbit 5ms delay 0.00000% loss) (100.00
Mbit 5ms delay 0.00000% loss) (h3, s1) (100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 5ms delay 0.0
0000% loss) (h4, s2) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us) h3 (cfs 150000/100000us) h4 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 0 delay
0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 5ms delay 0.00000% loss) (100.00Mbit 0
delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s1-eth2
h4 h4-eth0:s2-eth2
Testing bandwidth between h1 and h4 under TCP cubic
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['69.0 Mbits/sec', '69.2 Mbits/sec']
*** Iperf: testing TCP bandwidth between h3 and h4
*** Results: ['71.8 Mbits/sec', '73.0 Mbits/sec']
The bandwidth between h1 and h2 is: 69.2 Mbits/sec
The bandwidth between h3 and h4 is: 73.0 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 5 links
.....
*** Stopping 2 switches
s1 s2
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

We can know that the bandwidth between h1 and h2 is 69.2 Mbits/sec, while the bandwidth between h3 and h4 is 73.0 Mbits/sec, which is much fair than Reno.

Now, we consider the influence of packet loss. We set the link loss rate of (h3, s1) to 0.1%, and set the link loss rate of (h4, s2) to 0.1%. The result is:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_4_cubic_loss.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (h1, s1) (100.00Mbit 0 delay 0.
00000% loss) (100.00Mbit 0 delay 0.00000% loss) (h2, s2) (100.00Mbit 0 delay 0.10000% loss) (100.00Mb
it 0 delay 0.10000% loss) (h3, s1) (100.00Mbit 0 delay 0.10000% loss) (100.00Mbit 0 delay 0.10000% lo
ss) (h4, s2) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us) h3 (cfs 150000/100000us) h4 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.10000% loss) (100.00Mbit 0 delay 0.
00000% loss) (100.00Mbit 0 delay 0.00000% loss) (100.00Mbit 0 delay 0.10000% loss) (100.00Mbit 0 dela
y 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
h3 h3-eth0:s1-eth2
h4 h4-eth0:s2-eth2
Testing bandwidth between h1 and h4 under TCP cubic
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['60.9 Mbits/sec', '61.0 Mbits/sec']
*** Iperf: testing TCP bandwidth between h3 and h4
*** Results: ['64.7 Mbits/sec', '65.4 Mbits/sec']
The bandwidth between h1 and h2 is: 61.0 Mbits/sec
The bandwidth between h3 and h4 is: 65.4 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 5 links
.....
*** Stopping 2 switches
s1 s2
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
```

We can know that the bandwidth between h1 and h2 is 61.0 Mbits/sec, while the bandwidth between h3 and h4 is 65.4 Mbits/sec, which also contains a good fairness. The reason is that cubic function has a convex region which can explore more bandwidth in a rapid time. Additionally, cubic function has a flat region to carefully examine the CongWin, which maintains a rather high and stable bandwidth.

In conclusion, Cubic is much better than Reno. It's hardly surprising that it becomes the default TCP algorithm in GNU/Linux.

Reflection

At first, I set the iperf time to 10 sec, which is too short to get a stable bandwidth. In other words, there is a rather big change in bandwidth --- from 502 Mbits/sec to 352 Mbits/sec in Reno:

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_reno_std.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h2, s2) (1000.00Mbit 0 delay 0.00000% loss) (100
0.00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['496 Mbits/sec', '502 Mbits/sec']
502 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

```
littlestar@littlestar-VirtualBox:~/Documents/ComputerNetworking/mininet/labs/lab5/source$ sudo python
3 lab5_3_reno_std.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h1, s1) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (h2, s2) (1000.00Mbit 0 delay 0.00000% loss) (100
0.00Mbit 0 delay 0.00000% loss) (s1, s2)
*** Configuring hosts
h1 (cfs 150000/100000us) h2 (cfs 150000/100000us)
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...(1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss) (1000.00Mbit 0 delay
 0.00000% loss) (1000.00Mbit 0 delay 0.00000% loss)
Dumping host connections
h1 h1-eth0:s1-eth1
h2 h2-eth0:s2-eth1
Testing bandwidth between h1 and h4 under TCP reno
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['350 Mbits/sec', '352 Mbits/sec']
352 Mbits/sec
*** Stopping 1 controllers
c0
*** Stopping 3 links
...
*** Stopping 2 switches
s1 s2
*** Stopping 2 hosts
h1 h2
*** Done
```

So, I change the iperf time to 60 sec in order to get a stable bandwidth in a specific condition.