**Lab5：Testing TCP in Mininet**

叶增渝 519030910168

1.TCP bbr算法：

（1）特点：

TCP将拥塞控制和选择重传机制分离，分开判断。在传输过程中不断测量、估计最优带宽和延迟，计算出相应的cwnd，通过增加变量增益系数，控制发送端传输的增减速率，以解决发送端突发造成的网络排队问题。

（2）工作流程：  
a）startup：类似于慢启动，发送速率以指数方式增长，以期快速达到瓶颈，在等待3个RTT发现带宽未增加时，结束该状态进入drain阶段

b）drain：进入该阶段后后一个极小的增益系数增加，清空startup占用的大量网络缓存

c）probe\_bw：bbr的前述阶段一直在测量瓶颈带宽和最小RTT后，以一个稳定的匀速维护着网络状态，偶尔小幅提速、降速，但是依然是bbr的一个稳定状态，运行时间最长

d）probe\_rtt：在前述的三个阶段时，如果探测到的RTT不降

2.我采用了reno与bbr的congestion control算法，包装成为了bbr.py与reno.py两个源文件。采用sudo python3 reno.py与sudo python3 bbr.py两个命令来在Mininet中执行

[]中括号中左边表示发送端，右边表示接收端

3.改变每一个link上的带宽，保持各link的延迟为5ms，丢包率为0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 流量（单位Mbps） | bw=20 | bw=40 | bw=60 | bw=80 | bw=100 |
| reno | [18.4，25.0] | [34.3，41.9] | [52.8，60.7] | [69.2，77.6] | [84.0，91.9] |
| bbr | [19.0，26.8] | [34.7，36.6] | [47.6，50.2] | [60.6，65.9] | [77.4，81.8] |

改变每一个link上的延迟，保持各link的带宽为100，丢包率为0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 流量（单位Mbps） | delay=20ms | delay=50ms | delay=100ms | delay=200ms | delay=300ms |
| reno | [81.1，91.1] | [70.6，75.2] | [46.2，48.0] | [9.33，9.96] | [1.71，2.03] |
| bbr | [73.9，83.0] | [66.0，70.3] | [47.1，48.6] | [7.95，9.57] | [1.17，1.52] |

改变双switch相连的link上的丢包率，保持各link的带宽为100，延迟为20ms，其余loss rate为0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 流量（单位bps） | loss=0 | loss=0.01 | loss=0.02 | loss=0.05 |
| reno | [81.1M，91.1M] | [70.1M，78.8M] | [1.01M，1.21M] | [494K，2559K] |
| bbr | [73.9M，83.0M] | [66.5M，70.8M] | [60.3M，72.4M] | [690k，1.3M] |

总结：

（1）从上述实验的整体数据可以看出趋势：当仅有一个pair时，那么带宽越大，延迟越低，丢包率越小，均会导致网络吞吐量变大

（2）对比两种算法，可以发现在丢包率很低时，相同带宽与延迟，bbr的流量会略小于reno，但是一旦丢包率超过一个阈值，两者的流量都会迅速下降，但bbr的阈值更高

4.由于bbr算法会估计带宽与最短RTT，所以在单个iperf测试时已经设置好了相应的flow rule，所以我们对多pair使用bbr算法

考虑s1与s2之间的连接丢包率为0.1%，延迟200ms，带宽100，其余连接丢包率为0，延迟5ms，带宽100。那么显而易见，由于参数与所有数据都需要通过s1-s2连接，这个连接为bottleneck

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 流量（单位Mbps） | h1-h2 pair | h3-h4 pair | h5-h6 pair | h7-h8 pair | h9-h10 pair |
| bbr | [3.36，5.91] | [2.06，9.96] | [3.53，19.3] | [2.17，8.33] | [3.34，20.8] |

可以看到由于bottleneck存在，导致相比单pair，sender端对每个pair的吞吐量不算大，近乎平分了瓶颈的带宽