网络实验报告

雷正宇 2016K8009909005

实验内容

根据提供的脚本，重现如下结果：

h1(发送方)在对h2进行iperf的同时，测量h1的拥塞窗口值、r1-eth1的队列长度、h1与h2间的往返延迟

变化r1-eth1的队列大小，考察其对iperf吞吐率和ping延迟的影响

分别使用 codel, red, taildrop 三种方法解决 BufferBloat 问题

实验流程

利用脚本生成数据

编写 shell 脚本：

```
sudo python2 reproduce_bufferbloat.py -q 20
sudo python2 reproduce_bufferbloat.py -q 50
sudo python2 reproduce_bufferbloat.py -q 100
sudo python2 reproduce_bufferbloat.py -q 150
sudo python2 mitigate_bufferbloat.py -a taildrop
sudo python2 mitigate_bufferbloat.py -a red
sudo python2 mitigate_bufferbloat.py -a codel
```

绘制图表

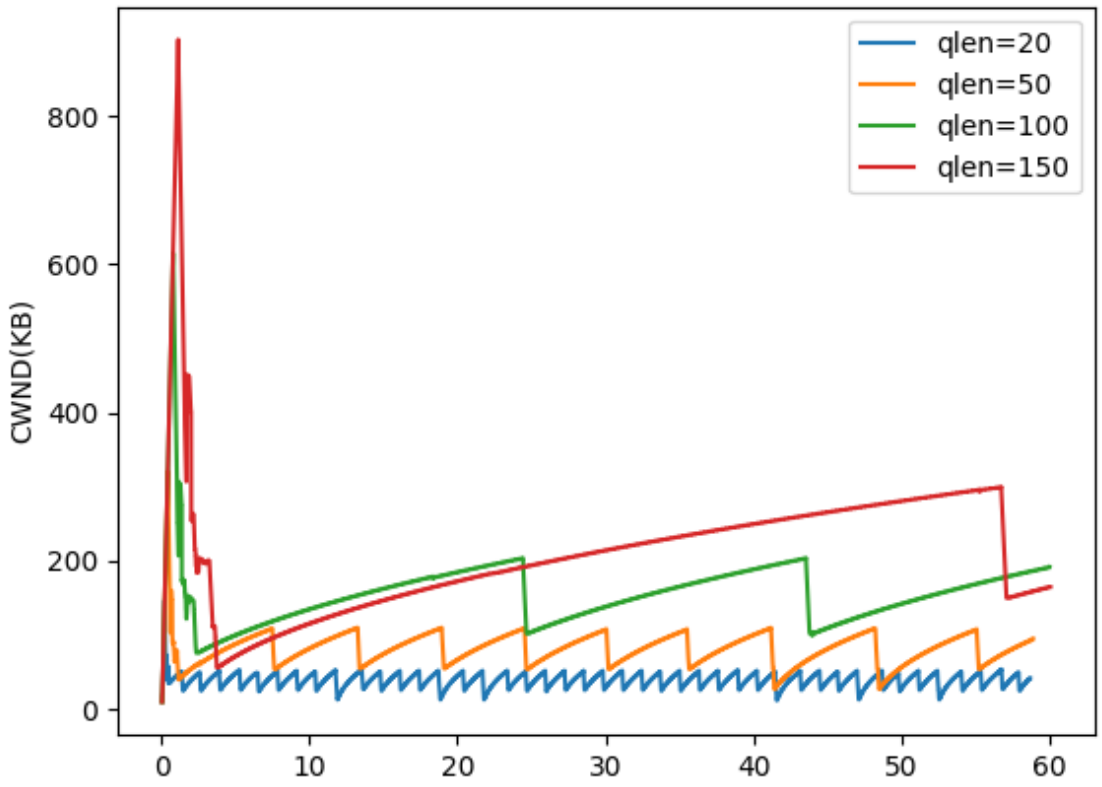
在本实验中，我使用 python3-matplotlib.pyplot 绘制对应图表，脚本如下：

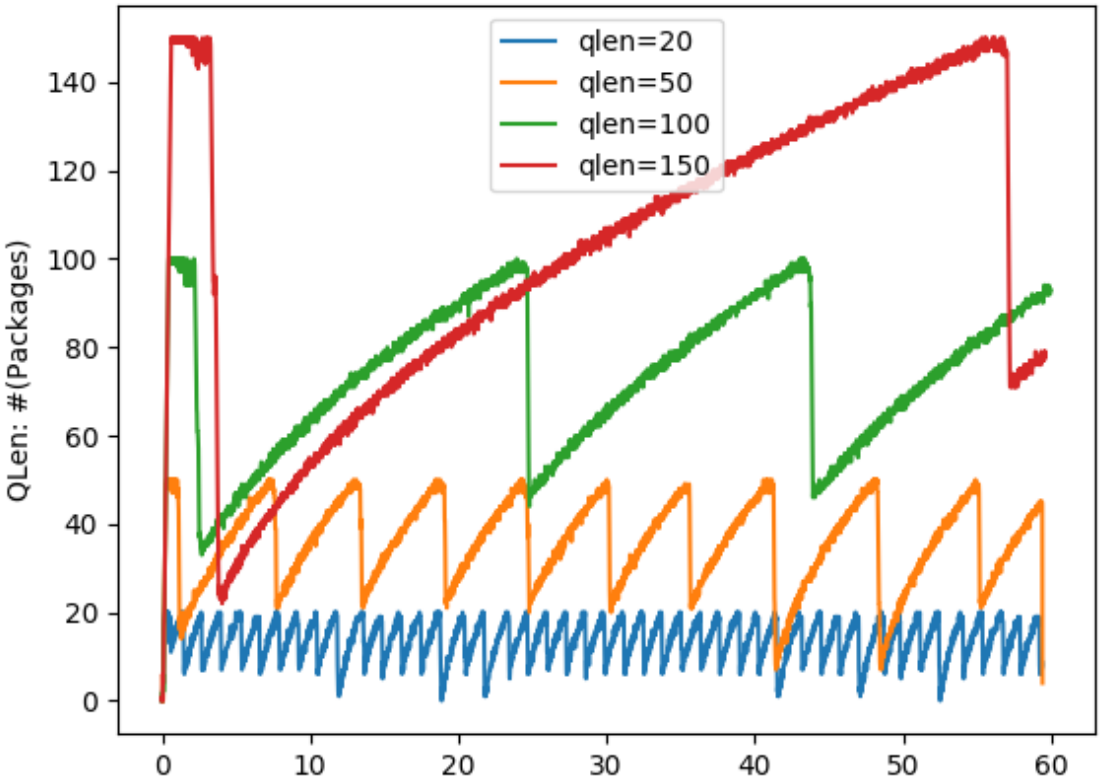
```
import matplotlib.pyplot as plt

qlen_list = [20, 50, 100, 150]

for qlen in qlen_list:
    with open('qlen-' + str(qlen) + '/cwnd.txt') as file:
        line = file.readline()
        x, y = [], []
        while line:
            if line.split()[1].split(':')[0] == '10.0.1.11':
                y.append(int(line.split()[6]))
                x.append(float(line.split()[0]))
            line = file.readline()
```

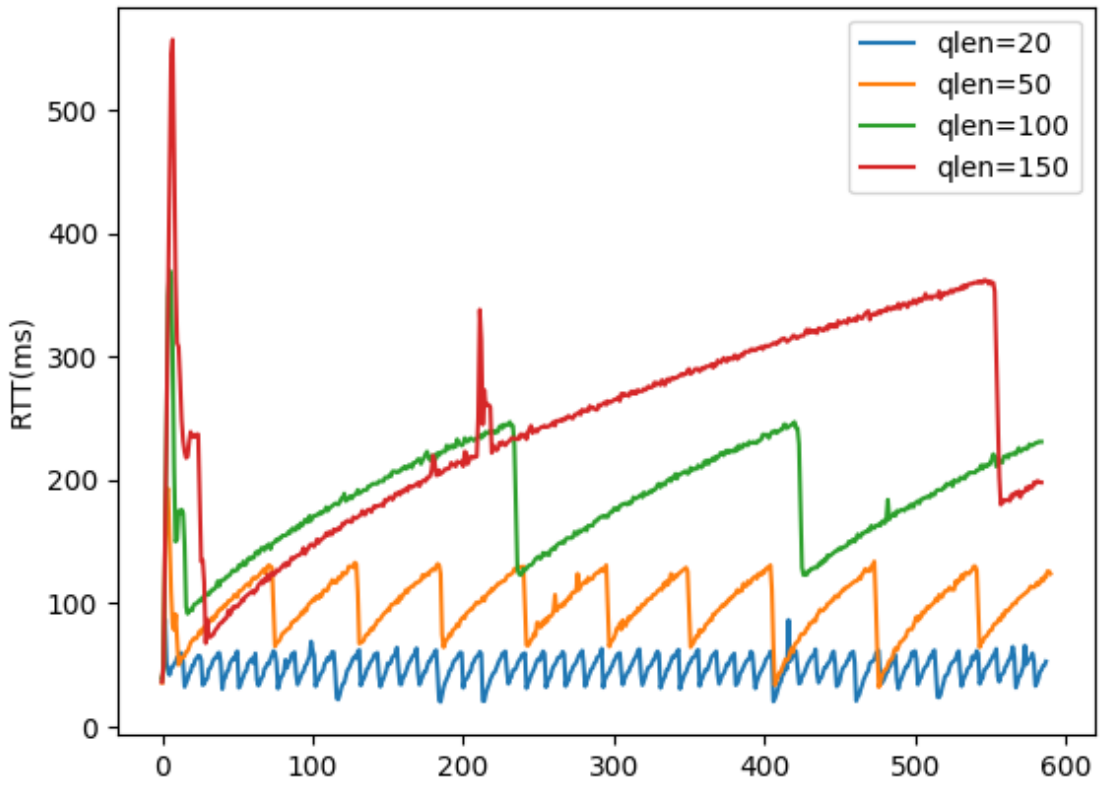
```
        plt.plot(x, y, label='qlen=' + str(qlen))

plt.ylabel('CWND(KB)')
plt.legend()
plt.savefig('qlen-crowd.jpg')

for qlen in qlen_list:
    with open('qlen-' + str(qlen) + '/qlen.txt') as file:
        line = file.readline()
        x, y = [], []
        while line:
            if len(line.split()) > 1:
                x.append(float(line.split()[0].split(',')[0]))
                y.append(int(line.split()[1]))
            line = file.readline()

        for i in range(1, len(x)):
            x[i] = x[i] - x[0]
        x[0] = 0.0
        plt.plot(x, y, label='qlen=' + str(qlen))

plt.ylabel('QLen: #(Packages)')
plt.legend()
plt.savefig('qlen-QLen.jpg')

for qlen in qlen_list:
    with open('qlen-' + str(qlen) + '/ping.txt') as file:
        line = file.readline()
        line = file.readline()
        x, y = [], []
        while line:
            y.append(float(line.split()[6].split('=')[1]))
            line = file.readline()
        x = list(range(len(y)))
        plt.plot(x, y, label='qlen=' + str(qlen))

plt.ylabel('RTT(ms)')
plt.legend()
plt.savefig('qlen-RTT.jpg')

scheme_list = ['taildrop', 'red', 'codel']
for scheme in scheme_list:
    with open(scheme + '/ping.txt') as file:
        line = file.readline()
        line = file.readline()
        x, y = [], []
        while line:
            y.append(float(line.split()[6].split('=')[1]))
            line = file.readline()
```

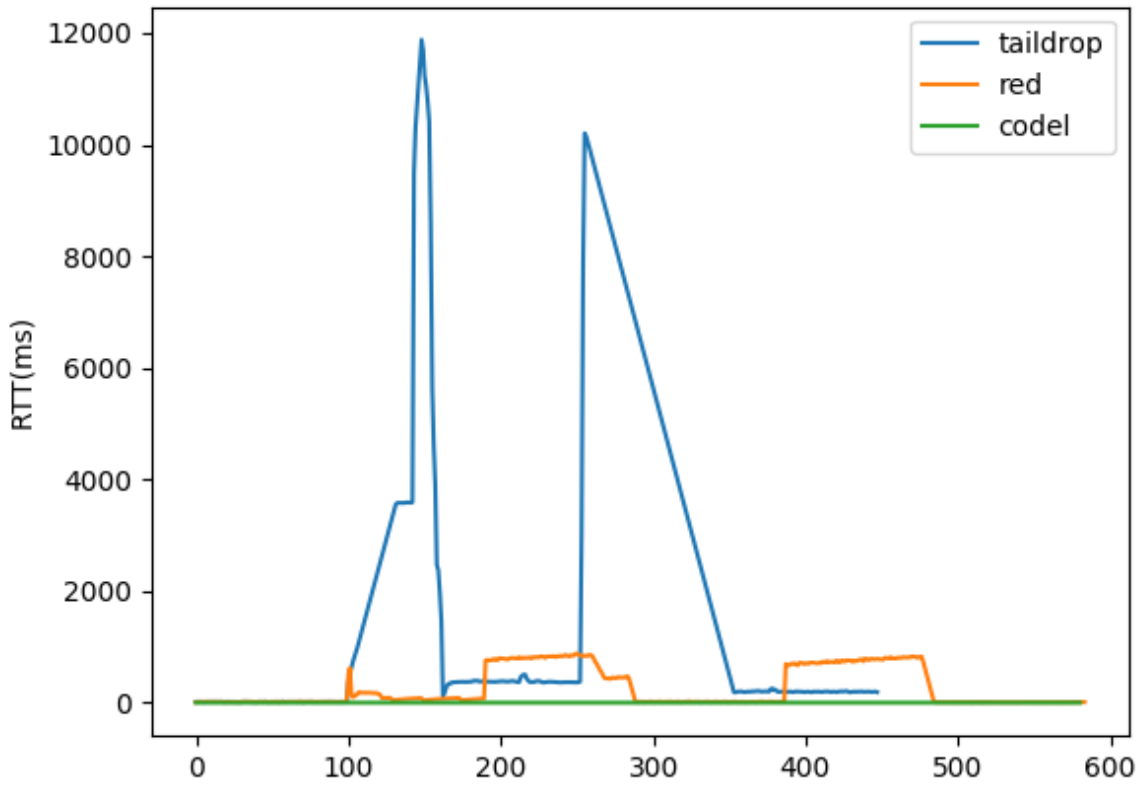
```
63          x = list(range(len(y)))
64          plt.plot(x, y, label=scheme)
65
66  plt.ylabel('RTT(ms)')
67  plt.legend()
68  plt.savefig('solve.jpg')
```

实验结果及分析




qlen=20
qlen=50
qlen=100
qlen=150
QLen: #(Packages)
0
20
40
60
80
100
120
140
0
10
20
30
40
50
60


qlen=20
qlen=50
qlen=100
qlen=150
RTT(ms)
0
100
200
300
400
500
0
100
200
300
400
500
600



TCP 的传输机制是，只要没有丢包，TCP 就会试图增大用色窗口，qlen 和 RTT 就跟着增大；遇到拥塞信号后，又回减小窗口，所以前三张图均呈现出周期性的锯齿形状。

taildrop 策略：

在某些情况下，在缓冲区填满之前丢弃一个分组的做法是有利的，这可以向发送方提供一个拥塞信号。但这样可能造成队列死锁，导致 TCP 全局同步：当缓冲区填满后，所有新到达的数据包被丢弃，所有发送方会同时降低发送速率；拥塞消除后，发送方又会同时增大发送速率，缓冲区又会很容易被填满，降低网络利用率。

RED：

RED 能够有效避免拥塞，但是参数设置很复杂，参数的细微变化经常会对网络性能造成很大影响。另外，平均队列长度经常会随着链接数的增加不断增大，造成传输时延抖动，引起网络不稳定。

Codel：

三种方法里最稳定，队列延迟最小。