数据包队列管理实验报告

张翔雨 2018K8009929035

一、实验题目：数据包队列管理实验

二、实验内容

- 重现BufferBloat问题
  - h1(发送方)在对h2进行 `iperf` 的同时，测量h1的拥塞窗口值(cwnd)、r1-eth1的队列长度(qlen)、h1与h2间的往返延迟(rtt)
  - 变化r1-eth1的队列大小，考察其对 `iperf` 吞吐率和上述三个指标的影响
- 解决BufferBloat问题
  - 根据附件材料中提供的脚本，重现实验结果

三、实验过程

1.重现BufferBloat问题

将脚本 `utils.py` 中的97行改为 `client = h1.cmd('iperf -c %s -t %d -i 0.1 > test.log | tee iperf_result.txt &' % (h2.IP(), duration+5))` ，每隔0.1s输出测试结果，分别在脚本所在目录下执行如下三条命令：

- `sudo python reproduce_bufferbloat.py -q 10`
- `sudo python reproduce_bufferbloat.py -q 100`
- `sudo python reproduce_bufferbloat.py -q 200`

生成了最大队列大小分别为10、100、200的实验数据。 `iperf` 报告需要每次执行后单独备份，cwnd、qlen、rtt的数据则在对应生成的文件夹下。

2.解决BufferBloat问题

分别在脚本所在目录下执行如下三条命令：

- `sudo python mitigate_bufferbloat.py -a taildrop`
- `sudo python mitigate_bufferbloat.py -a red`

- `sudo python mitigate_bufferbloat.py -a codel`

得到分别采用三种解决算法的rtt结果，在对应算法名的目录下。

四、实验结果及分析

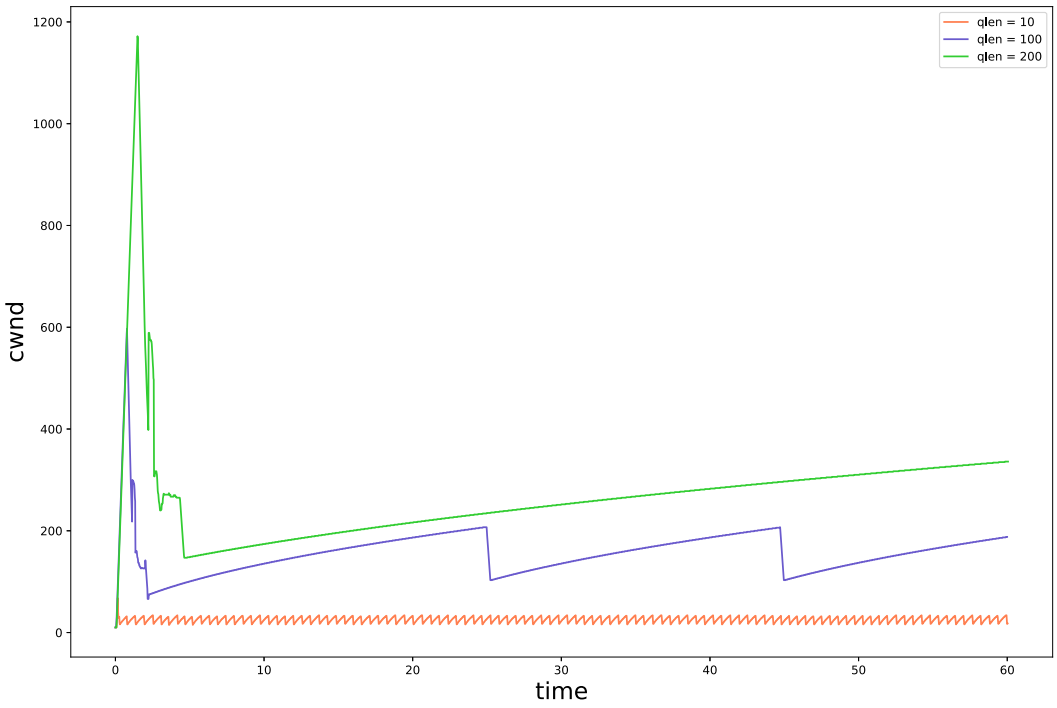
1.数据处理

在实验主目录下执行 `data_cut.py`，对实验获得的数据进行提取处理，保存在以 `\t` 作为分隔符的 `csv` 文件中。具体数据拼接方法较为繁琐，详见代码文件。

2.重现BufferBloat问题

（1）CWND

在实验主目录下执行 `drawing_cwnd.py`，得到绘制的图形，其中横坐标是所有的时间与第一个时间的差值，总区间在0~60左右，保证三条折线横坐标相同，纵坐标为得到的cwnd数据。



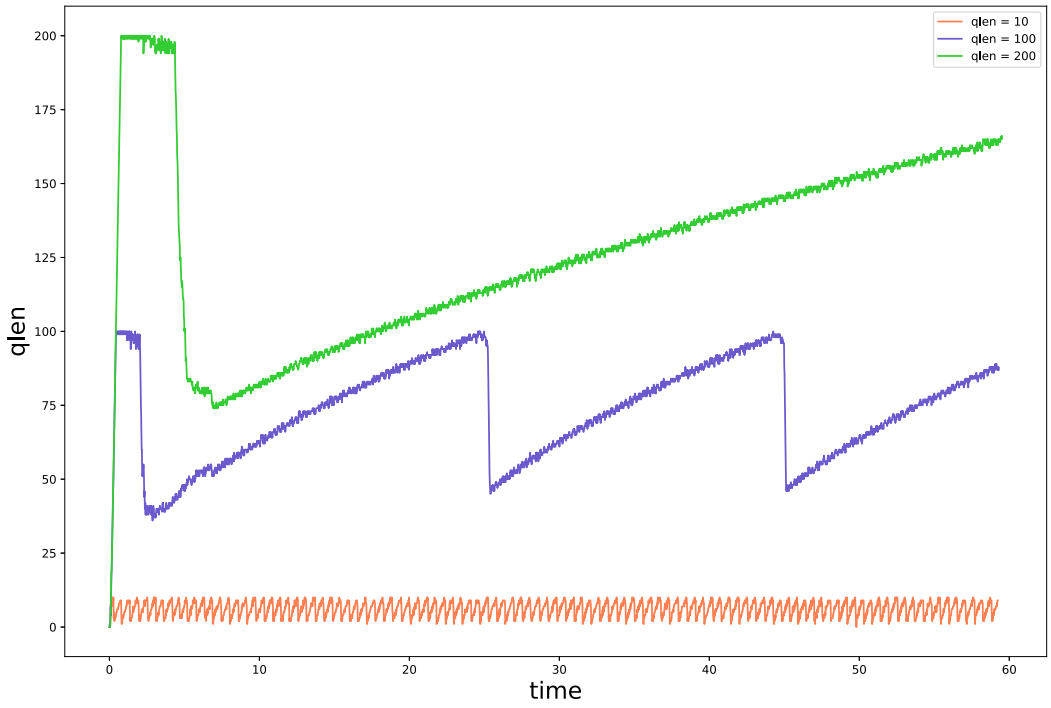
cwnd-time折线图

可以看到当maxq=100或200时折线都会在开端迅速增长至峰值，然后迅速回落，该峰值约为设定的最大队列大小的6倍。

maxq=100的情况在回落后表现出了与qlen=10的折线相同的震荡趋势，且qlen=100的震荡周期明显较长，猜测qlen=200的折线也会发生震荡，因为测试时间较短而未达到震荡周期。而震荡时的拥塞窗口值（cwnd）约和最大队列大小相同。

（2）QLEN

在实验主目录下执行`drawing_qlen.py`，得到绘制的图形，其中横坐标是所有的时间与第一个时间的差值，总区间在0~60左右，保证三条折线横坐标相同，纵坐标为得到的qlen数据。

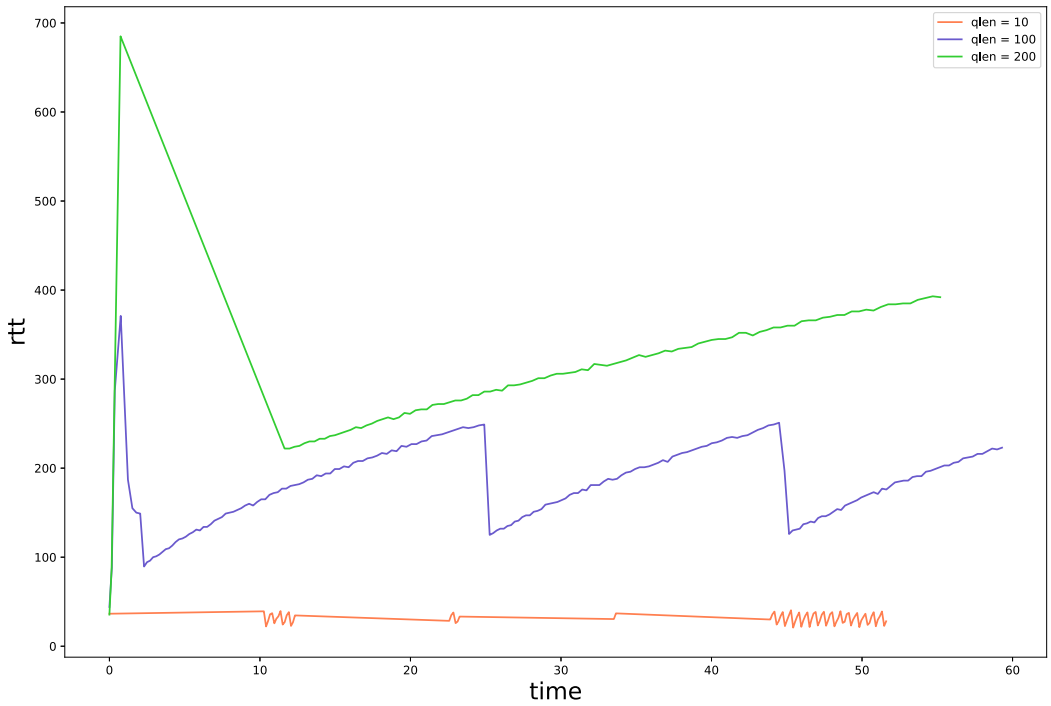


qlen-time折线图

三条曲线的峰值皆为设定的最大队列大小。当曲线达到峰值后，队列变满，将丢弃新收到的包。随着发送速率的降低，队列大小也将快速下降，并保持在最大队列大小的37.5%~100%之间震荡，做周期性变化。

（3）RTT

在实验主目录下执行`drawing_rtt.py`，得到绘制的图形，其中横坐标是所有的时间与第一个时间的差值，总区间在0~60左右，保证三条折线横坐标相同，纵坐标为得到的rtt数据。



rtt-time折线图

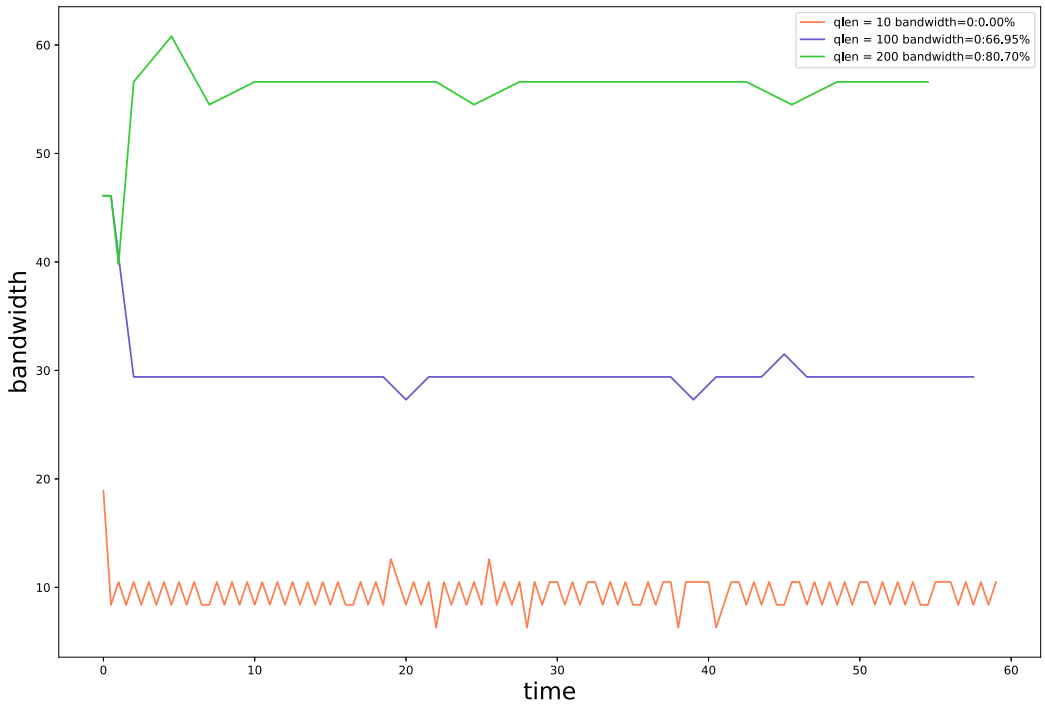
可以观察到rtt-time的曲线图与qlen-time图的趋势基本相似，表明往返延迟随着队列大小产生着正相关的变化。

往返延时的峰值约为最大队列大小的3.5倍，同时当最大队列越小时，往返延迟的周期性变化越不明显。

（4）iperf吞吐率

ubuntu 16.04 的结果

在这里使用带宽代表网络中吞吐率的变化，在实验主目录下执行 `drawing_bandwidth.py`，得到绘制的图形，其中横坐标是所有的时间与第一个时间的差值，总区间在0~60左右，保证三条折线横坐标相同，纵坐标为得到的bandwidth数据。实验结果去除了带宽为0的数据点



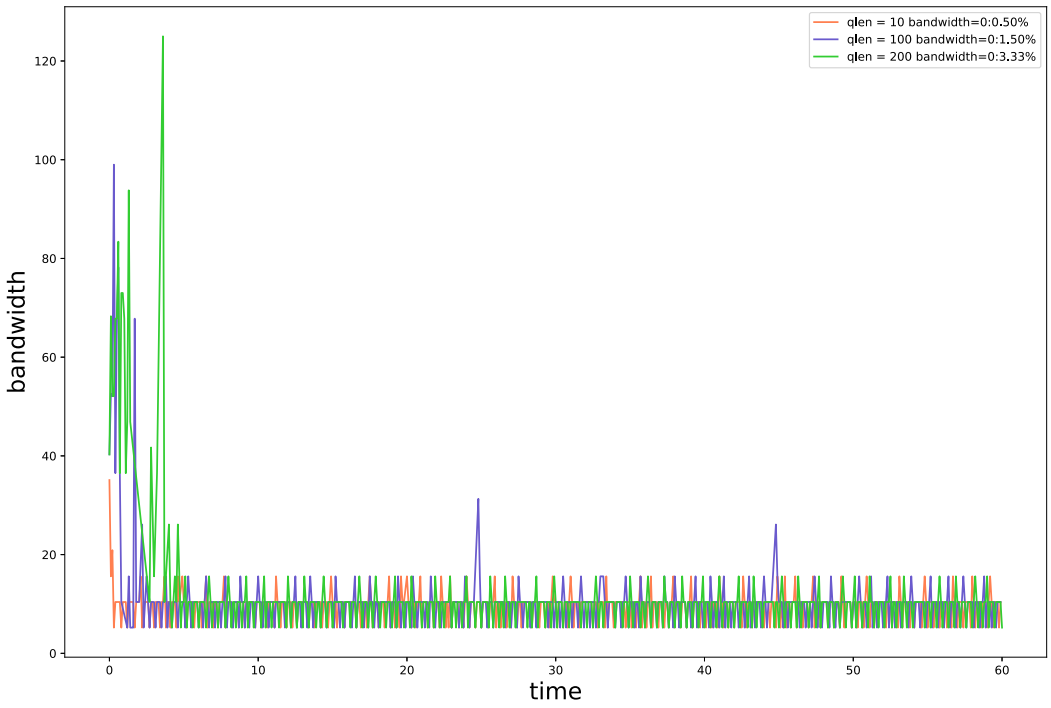
bandwidth-time折线图-ubuntu 16.04

可以看到平均带宽与最大队列大小基本成正相关关系。

处理实验数据时发现在qlen = 100后带宽中开始出现为0 的情况，统计了带宽为0的情况所占的比例，展示在折线图的图例中qlen=10、100、200对应的比例分别为0.00%，66.95%，80.70%，可以发现随着最大队列长度的增长，这种情况出现的比例也在显著增多，在maxq=100时现象变得明显。

ubuntu 20.04 的结果

在实验中发现ubuntu的版本不同，得到的实验结果差异较大，故两者都进行展示。



bandwidth-time折线图-ubuntu 20.04

可以看到除了开端波动较大，之后的三条线的实验数据成相同的周期性变化，在10附近震荡。而数据为0的比例也少了很多。此时与16.04的情况进行对比发现两点不同:

- 带宽不再与队列大小成明显关系。由于实验中设定的带宽为10，而在16.04中却产生了maxq=100,200时带宽长期超过10的情况，故认为16.04的数据存在问题。
- 带宽为0的情况显著减少。20.04下测出的数据更加稳定，16.04则产生了大量的带宽缺少。

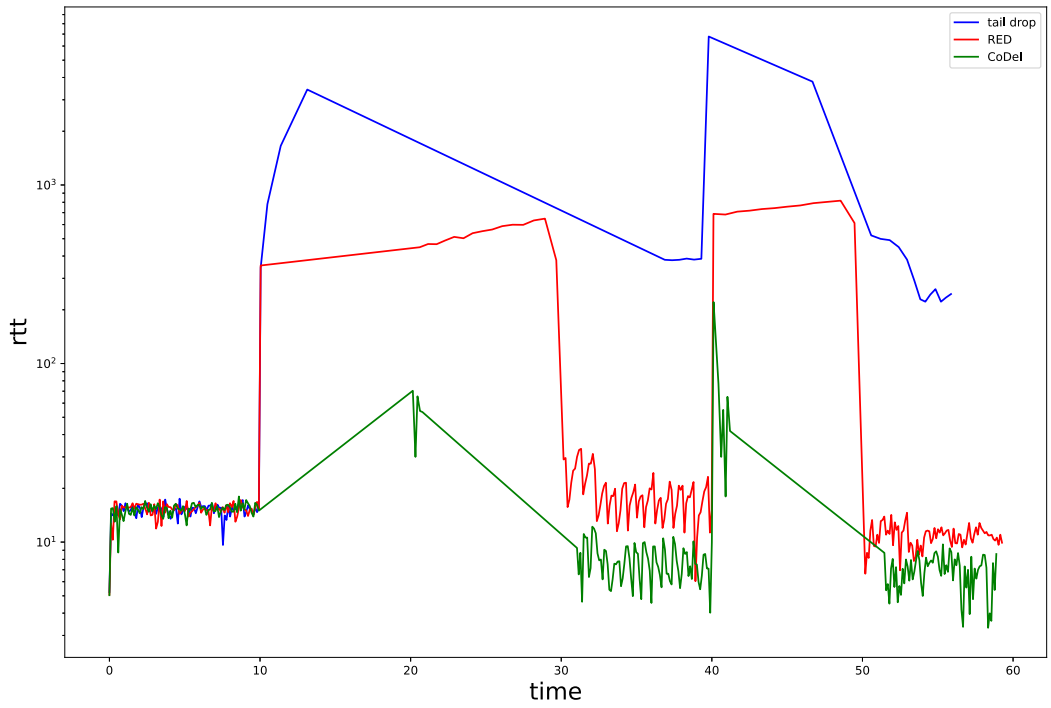
上述两种情况的原因也可能与iperf的版本有关：16.04中使用的iperf版本为2.0.5，而20.04中则使用2.0.13版本，差异比较大。

（5）总结

可以较为明显的看到随着最大队列长度的增加，BufferBloat的问题变得更加明显。TCP传输机制控制在没有丢包的情况下，只要不产生拥塞就会不断增加窗口大小以增加吞吐率。而当tail drop启动时，此时网络中已存在非常大的延迟，产生BufferBloat问题，而最大队列长度越长，BufferBloat问题也就越严重，但同时较小的队列长度会导致难以容忍高并发数据包，产生TCP Incast问题

3.解决BufferBloat问题

在实验主目录下执行`drawing_rtt_algo.py`，得到绘制的图形，其中横坐标是所有的时间与第一个时间的差值，总区间在0~60左右，保证三条折线横坐标相同，纵坐标使用log坐标，为得到的rtt数据。下图展示了三种算法处理问题的结果。



三种算法下的rtt-time折线图

从解决问题的效果上看，tail drop的效果表现最差，延时很高，CoDeL算法解决问题的效果最好，延时较其他两种最低。

同时可以观察到tail drop的折线出现明显的尖端现象，这是仿真环境的差异导致的。

五、思考题

1.调研分析两种新型拥塞控制机制（BBR [Cardwell2016], HPCC [Li2019]），阐述其是如何解决Bufferbloat问题的。

BBR：

1. 即时速率的计算：计算一个即时的带宽bw，该带宽是bbr一切计算的基准，bbr将会根据当前的即时带宽以及其所处的pipe状态来计算pacing rate以及cwnd。在bbr运行过程中，系统会跟踪当前为止最大的即时带宽。
2. RTT的跟踪：在bbr运行过程中，系统会跟踪当前为止最小RTT。

3. bbr pipe状态机的维持：bbr算法根据互联网的拥塞行为有针对性地定义了4中状态，即STARTUP，DRAIN，PROBE_BW，PROBE_RTT。bbr通过对上述计算的即时带宽bw以及rtt的持续观察，在这4个状态之间自由切换，相比之前的所有拥塞控制算法，其革命性的改进在于bbr拥塞算法不再跟踪系统的TCP拥塞状态机，而旨在用统一的方式来应对pacing rate和cwnd的计算，不管当前TCP是处在Open状态还是处在Disorder状态，抑或已经在Recovery状态，换句话说，bbr算法感觉不到丢包，它能看到的就是bw和rtt。

HPCC：

HPCC 是在高性能的云网络环境下，对现有的拥塞控制的一种替代方案。它可让数据中心网络中的报文稳定的、以微秒级的延迟传输。HPCC 则创新性地运用了最新网络设备提供的细粒度负载信息而全新设计了拥塞控制算法。HPCC 的核心理念是利用精确链路负载信息直接计算合适的发送速率，而不是像现有的 TCP 和 RDMA 拥塞控制算法那样迭代探索合适的速率；HPCC 速率更新由数据包的 ACK 驱动，而不是像 DCQCN 那样靠定时器驱动。

七、实验总结

本次实验表面上看没有什么代码量，但其实数据处理与绘图还是书写了大量的代码，也比较耗时。通过本次实验对BufferBloat的理解加深了很多。在iperf吞吐率变化的测量中，实验数据误差较大，且出现了数据完全不同的情况，还是比较令人困惑的。

八、提交文件说明

- `data_cut.py`:数据处理脚本
- `drawing_rtt.py`，`drawing_cwnd.py`，`drawing_qlen.py`，`drawing_bandwidth.py`:重现BufferBloat问题时的绘图脚本
- `drawing_rtt_algo.py`:对比三种算法解决BufferBloat问题的绘图脚本
- 其余为实验脚本及产生的实验数据