计算机网络实验 2 报告

学号　2017K8009929059
姓名　於修远

BufferBloat 问题重现

一、实验内容

- 运行` reproduce_bufferbloat.py `脚本，测算队列大小一定时，iperf 吞吐率和 ping 延迟在数据传输过程中的变化。
- 改变队列大小，比较其对 iperf 吞吐率和 ping 延迟的影响。

二、实验流程

（一）网络数据测量

1、 执行` sudo reproduce_bufferbloat.py -q 100 `，测量队列长度上限为 100 时，间隔 0.1s 执行一次、持续 60s 的 ping 操作得到的延迟值和执行 60s 的 iperf 得到的队列长度、拥塞窗口测量值。

2、 改变-q 参数的值，测量不同最大队列长度下的 iperf 和 ping 结果。

（二）数据处理分析

1、 对 ping 结果，用 icmp 包的序号除以 10 作为横坐标的时间轴（因为每 0.1s 执行一次 ping），测得的延迟作为纵坐标的 RTT 轴，绘制图像。

2、 对 qlen 结果，把第一列的所有值减去第一行第一列的值，得到测试过程的相对时间，作为横坐标的时间轴；第二列的即时队列长度作为纵坐标，绘制图像。

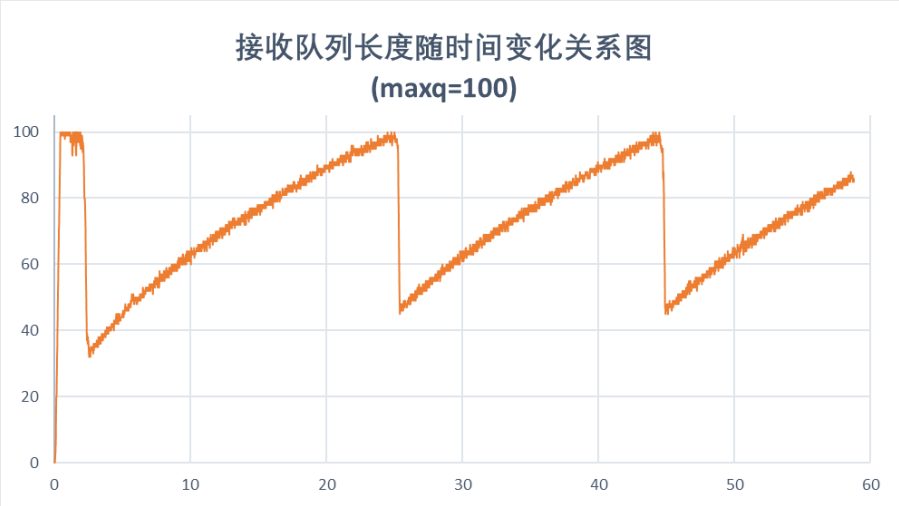
3、 对 cwnd 结果，保留第一（时间）、二（源地址：端口）、三（目的地址：端口）、七（拥塞窗口）数据，再筛选出所有目的地址：端口为 10.0.2.22:5001 的条目，以时间为横坐标，拥塞窗口为纵坐标，绘制图像。

三、结果分析

（一）问题复现

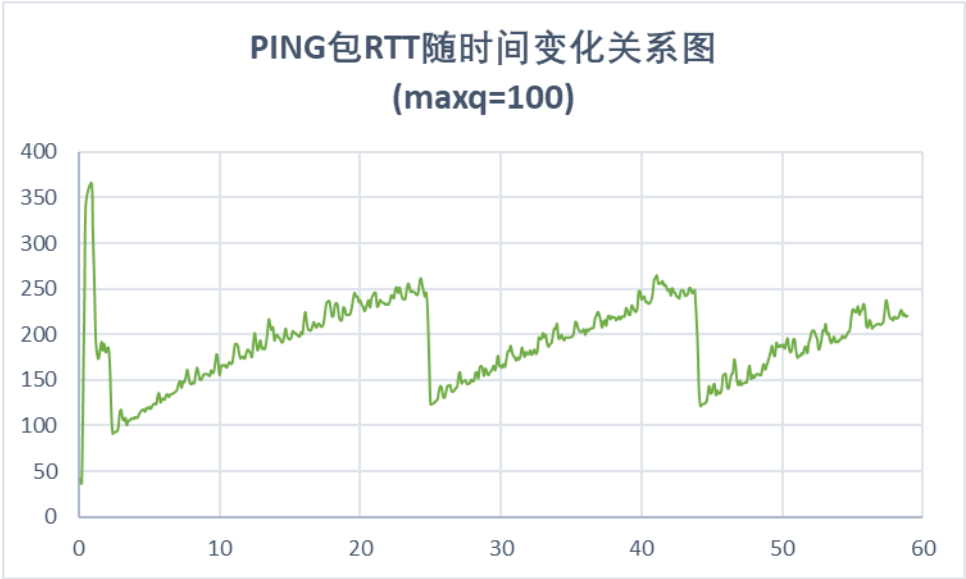
1、即时队列长度

如下图所示为即时队列长度随时间变化的曲线。可以看到曲线的上界为 100，即预设的最大队列长度。测试刚开始时队列迅速达到满载，当队列满时，由于默认使用 tail drop 方法处理，直接丢弃新接收的包，所以由于发送速率骤降，使得队列长度降低至最大队列长度的 40%左右，如是往复。



2、网络延时大小

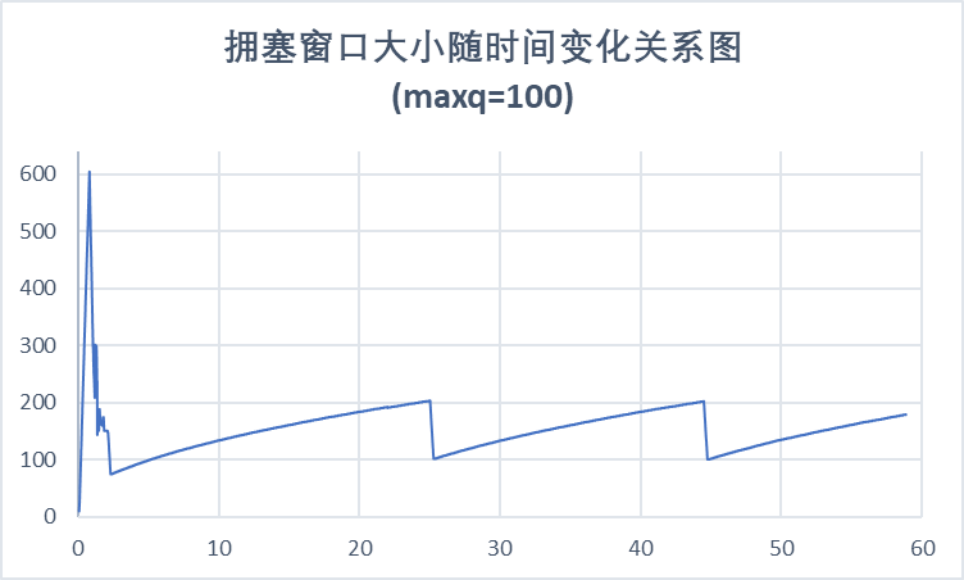
如下图所示为 PING 包的 RTT 随时间变化的曲线。测试刚开始时，由于接收队列的快速增长，网络延时也急剧增加，随后也随着队列缩短而减少。之后的时间内，延时随着队列长度的变化而变化，基本呈线性关系，但波动更为明显。



3、拥塞窗口大小

如下图所示为拥塞窗口大小随时间变化的曲线。测试刚开始时，拥塞窗口迅速增长达到约 600 的峰值，接着迅速回落到 100 左右的水准。此后，在图像上拥塞窗口也表现出与队列长度相似的变化周期和趋势，随

着收包而增长，因为丢包而骤降。



4、三项参数关系分析

首先，接收队列的长度部分受到拥塞窗口大小的影响，这是因为其长度直接取决于发送方的发包速率，而发送方发包速率又同时受到滑动窗口和拥塞窗口大小制约。所以理论上说，拥塞窗口大时可以达到更大的发送速率，接收队列增长也就越快。但是事实上从图像中可以看到随着拥塞窗口增大，接收队列增速反而减缓，可以推测是由于时延增大引起的滑动窗口首先造成的。

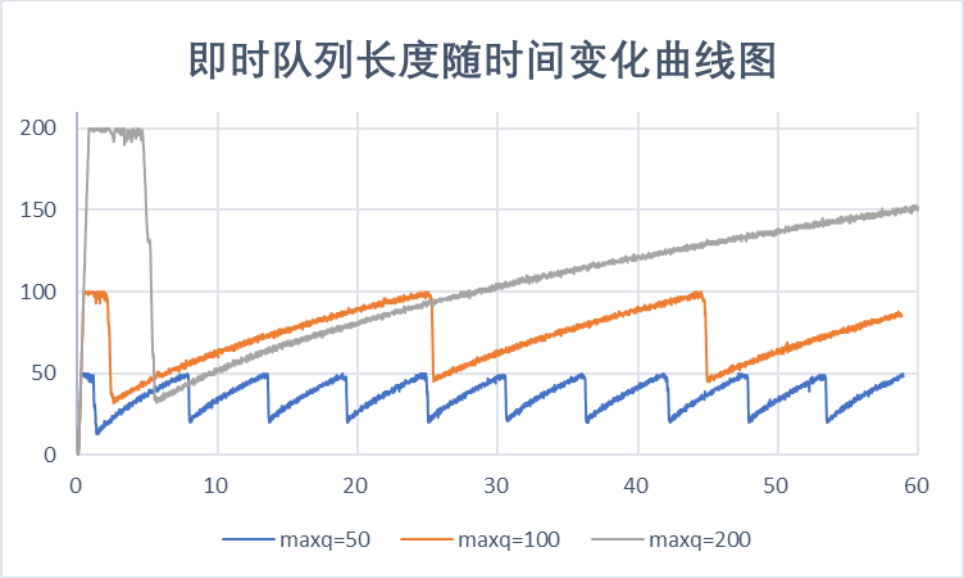
其次，随着接收队列增长，数据包的延时会线性增长。当队列长度很长时，ping 延时也会相应增长，这也就是 BufferBloat 问题的产生。当 tail drop 机制触发，队列瞬间缩短，延时也相应大幅减少。

最后，由于 ping 包返回延时增加，相当于接收返回 ping 包的速率下降，随接收返回包而增大的拥塞窗口的增长速度（图像的导数）也放缓。当队列长度达到上限时开始丢弃新接收的包，这会导致拥塞窗口因为丢包而快速减小。三者就这样互相影响互相关联。

（二）不同队列长度下的指标差异

1、即时队列长度

如下图所示，分别设定最大队列长度为 50、100、200，测得的 60s 内即时队列长度的变化。可以看到，改变最大队列长度，不影响时间曲线总体的变化趋势。由于短队列更容易被填满，所以最大队列长度越短，变化周期越短。同时，虽然每次的丢包数量更少，但短队列触发丢包频率会高很多，每次丢包后都需要一个传输增速的过程。在实际传输过程中，这可能反过来增加总的传输时长。



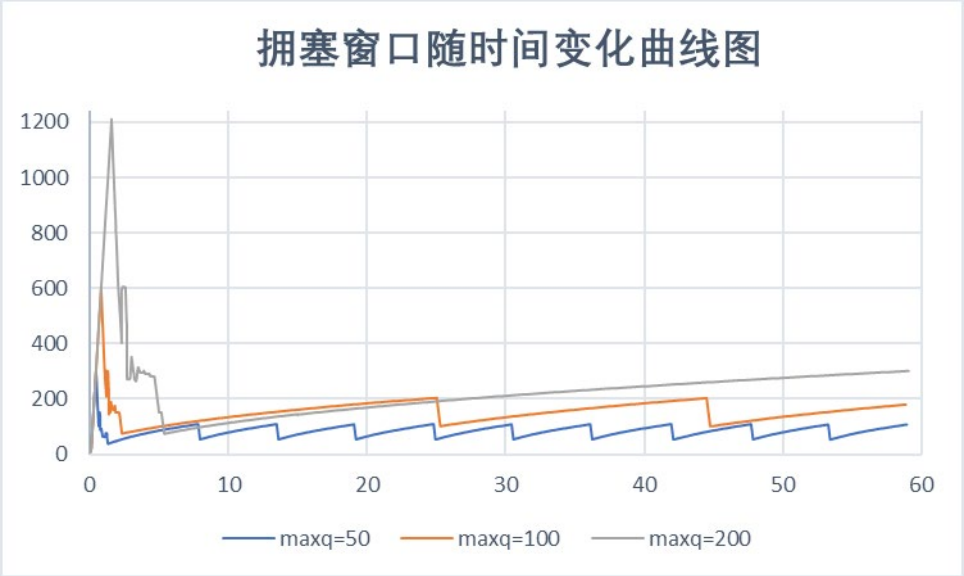
2、网络延时大小

如下图所示，分别设定最大队列长度为 50、100、200，测得的 60s 内 PING 延时的变化。最大队列长度决定了延时的上限，且最大队列长度大的情况下，高延时也会维持更长的时间。如图中显示的，最大队列长度为 200 时，可能平均会有 70%的时间处于 250ms 以上的延时。如果最大队列长度进一步增大，这一延时的数值和持续时间都会随之进一步增加，对网络性能影响将会十分明显。从数据条目上也可以看到，最大队列长度为 50、100、200 的情况下测得的 1min 内 PING 包返回数分别为 595、585、581，即大队列可能会导致更多包的超时或直接丢弃（主要集中在传输起始阶段）。

3、拥塞窗口大小

如下图所示，分别设定最大队列长度为 50、100、200，测得的 60s 内拥塞窗口的变化。改变最大队列长度，对拥塞窗口大小的影响是最为间接的。只是随着最大队列长度的增加，拥塞窗口可以达到一个更大的值，这一峰值和最大队列长度基本呈线性关系。状态稳定后，拥塞窗口的变化趋势维持同一规律，只存在幅度和

周期的差异。



（三）分析总结

综上所述可以看到，在使用 tail drop 的情况下，网络会产生 BufferBloat 问题，而且这一问题会随着最大接收队列长度的增加而愈发严重。但一味减小最大队列长度也容易引起反复重启，影响传输效率，还有可能引发 TCP Incast 问题。所以在应用 tail drop 方式的前提下，应当选择一个长度适中的接收队列；又或者是改进 tail drop 方法，来尽量避免 BufferBloat 的发生。但无论采取何种方法，合适的队列长度都是有必要的，单纯的增加和减少都会引起性能的下降。

BufferBloat 问题解决

一、实验内容

- 运行` mitigate_bufferbloat.py `脚本，测算队列大小一定时，不同的队列管理机制下， ping 延迟在数据传输过程中的变化。
- 绘制对比图像

二、实验流程

（一）网络数据测量

1、 执行` sudo mitigate_bufferbloat.py -a taildrop `，测量用 taildrop 策略管理长度上限为 1000 的接收队列时，执行持续 60s 的间隔为 0.1s 的 ping 操作得到的延迟值。

2、 调整-a 的参数为 red 和 codel，分别测量 1min 内的延迟变化。

（二）数据处理分析

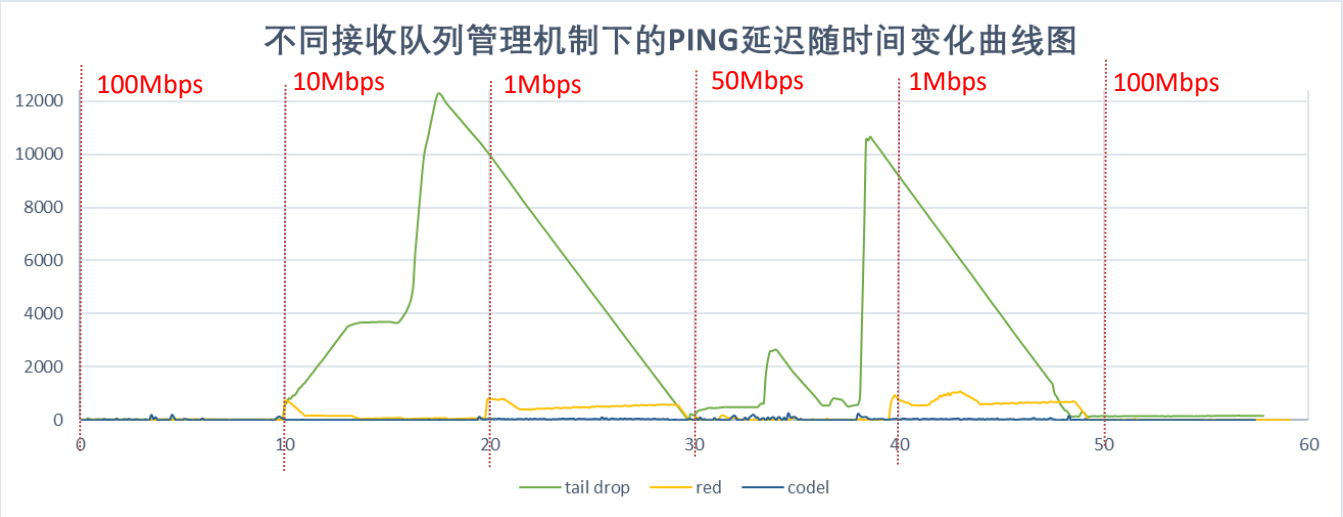
1、 保留所得记录表中的 icmp_seq 项和 time 项信息。

2、 用 icmp 包的序号除以 10 作为横坐标的时间轴（因为每 0.1s 执行一次 ping），测得的延迟作为纵坐标的 RTT 轴，整合三种策略的数据，绘制图像。

三、结果分析

（一）图像说明

如下图所示，为整合了三种队列管理机制的延时变化图像。



从图像中可以看到，tail drop 的队列管理策略在延时指标上要远远劣于 RED 策略和 CoDel 策略。在带宽突减时，PING 延时会急剧增加。检查数据也可以看到，在图中的线性下降部分，有十分严重的丢包情况，导致这些数据段只能用少数点进行拟合。这也是本图像与课件中图像存在差异的主要原因。

（二）结果分析

单纯从延时角度来看，CoDel 策略可以最大程度地规避 BufferBloat 问题。RED 策略在带宽较小的条件下性能会受到一定影响，但总体来看还可以接受。二者在延时指标上都要优于简单的 tail drop 策略。考虑到调参的难度和适用性以及实际测试得到的性能，CoDel 策略更胜一筹。但是由于没有考虑吞吐率相关的问题，所以不能断言 CoDel 策略在实际应用中更有效。