07-数据包队列管理实验报告

陈彦帆 2018K8009918002

1、实验内容

（1）根据附件材料中提供的脚本，重现 BufferBloat 问题：对于给定拓扑，变化中间路由器的最大队列长度，测量发送方拥塞窗口值(cwnd)、路由器队列长度(qlen)、rtt、bandwidth 随时间的变化，并绘图。

（2）根据附件材料中提供的脚本，给出三种策略（red,taildrop,codel）下 BufferBloat 测量的结果，并绘图。

（3）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。

2、实验流程

（1）重现 BufferBloat 问题：

修改 reproduce_bufferbloat.py 并执行，设置最大队列长度(maxqlen)为 20,50,100，运行 60s，测得在给定拓扑下发送方拥塞窗口值(cwnd)、路由器队列长度(qlen)、rtt、bandwidth 随时间的变化。

（2）解决 BufferBloat 问题:

修改 mitigate_bufferbloat.py 并执行，设置 maxqlen 为 1000，设置带宽变化为[100,10,1,50,1,100]，测量在三种策略（red,taildrop,codel）下，rtt 随时间的变化。

（3）用 matplotlib 绘图，其中（2）的图采用截断 y 轴。见 print_rep.py。

（4）完成调研题。

3、结果与分析

（1）重现 BufferBloat 问题

网络拓扑如图 1。变更 maxqlen，复现结果如图 2-5。

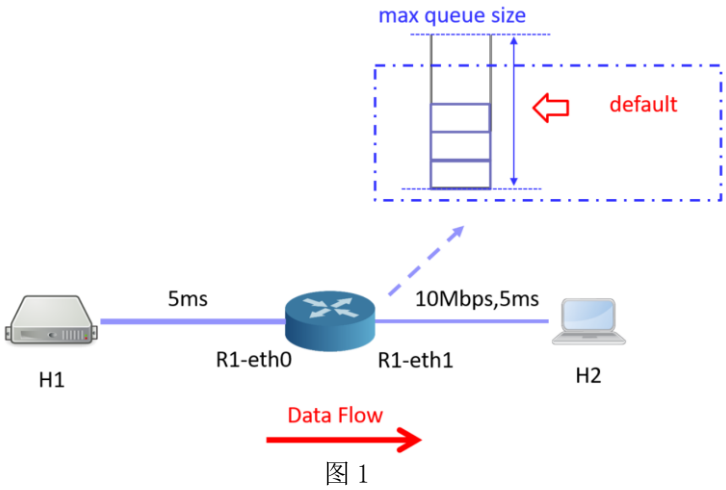

图 1

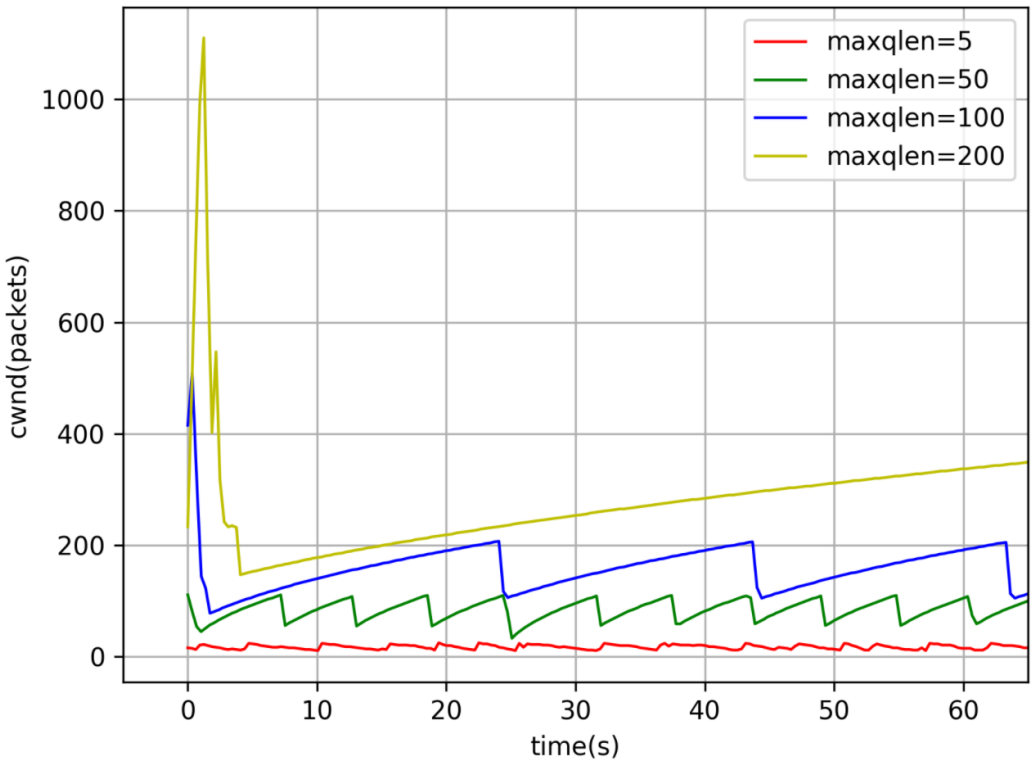

图 2 不同 maxqlen 大小下发送方拥塞窗口值(cwnd)随时间的变化

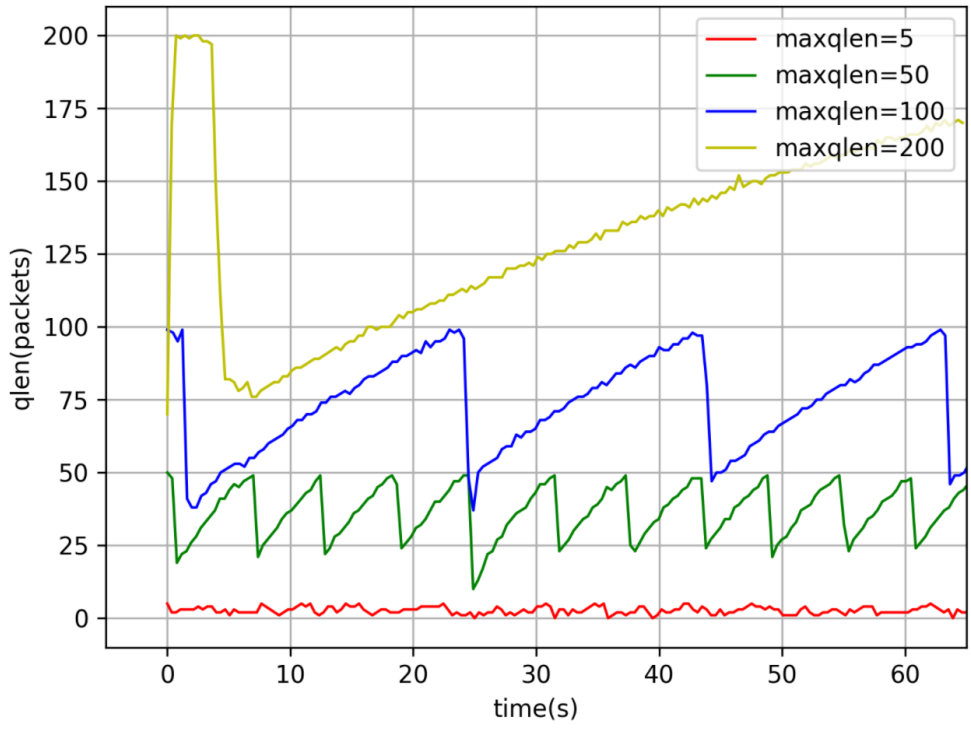

图 3 不同 maxqlen 大小下路由器队列长度(qlen)随时间的变化

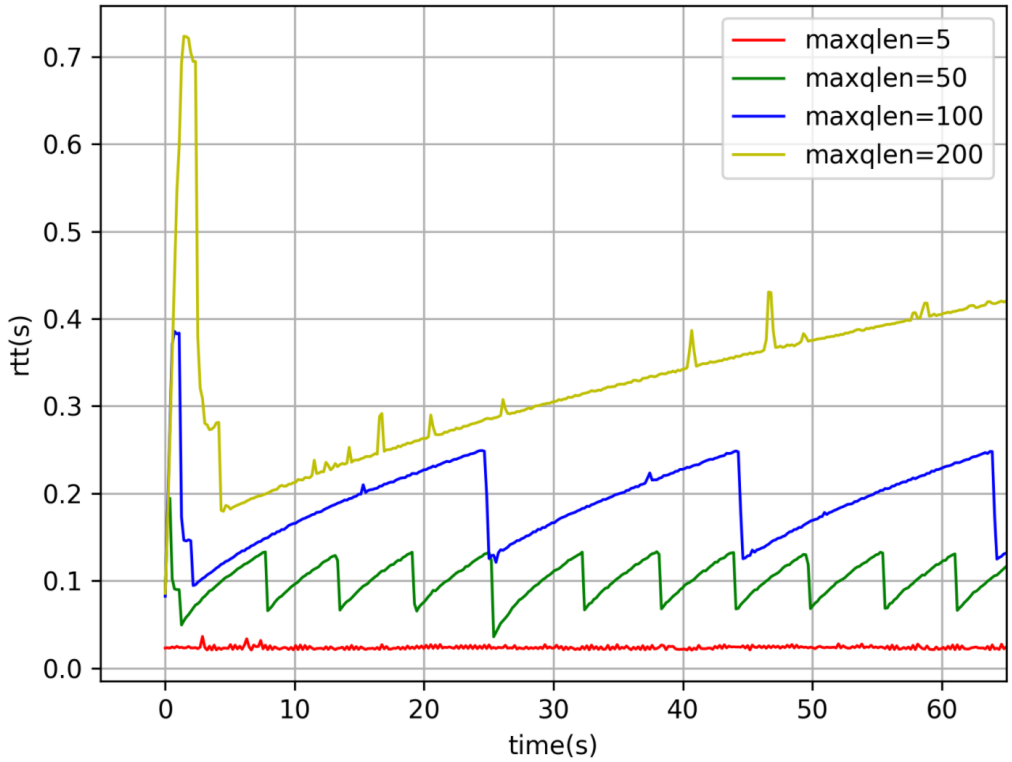

图 4 不同 maxqlen 大小下 rtt 随时间的变化

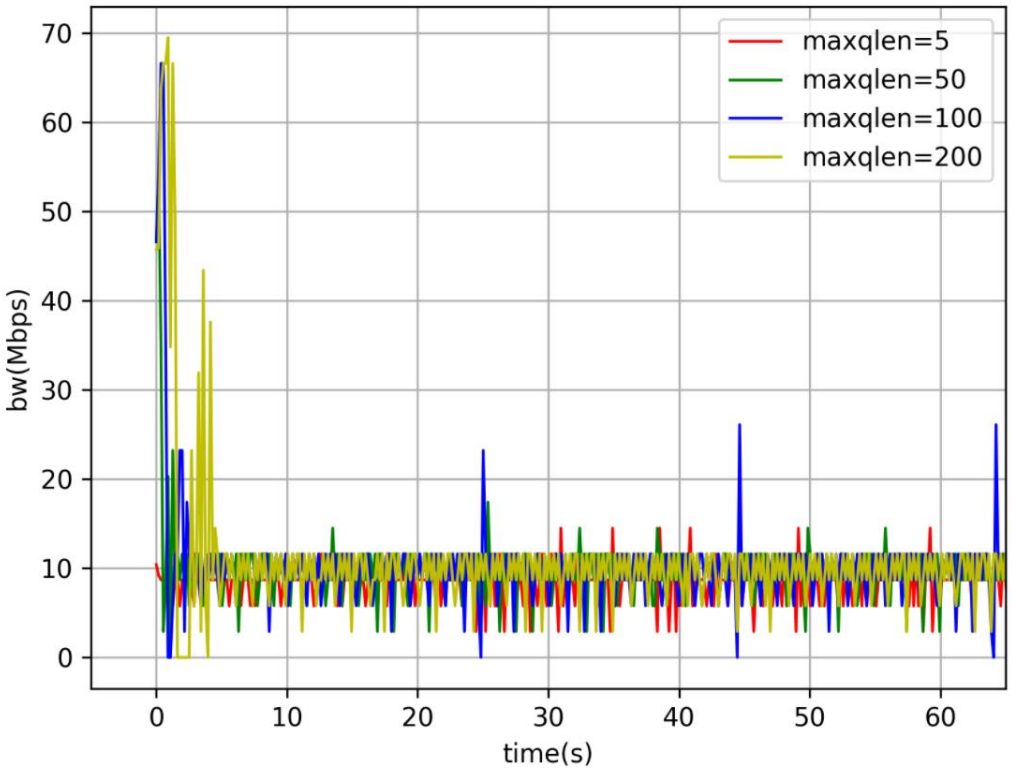

图 5 不同 maxqlen 大小下吞吐率随时间的变化

比较图 2，图 3，图 4，它们的行为都很相似。当 maxqlen 较小时（maxqlen=5），发送队列维持在一个较小的值，数据包滞留现象不明显，不产生 bufferbloat，rtt 也很低。当 maxqlen 增大到一定程度时(maxqlen=50,100,200)，一开始，由于瓶颈链路(r1-h2)的存在，从 h1 到达 r1 的数据包迅速填满 r1 的发送队列，造成高延迟(rtt)和丢包，TCP 拥塞控制机制迅速减小发送窗口，使 cwnd,qlen,rtt 均迅速降低，接着 TCP 缓慢增大发送窗口（cwnd），使 qlen 缓慢增大，rtt 也因为数据包在发送队列的滞留而增大。直到 qlen=maxqlen，又开始丢包，进入新一轮循环。

从图 5 看出，稳定以后，不同 maxqlen 的吞吐率均在瓶颈链路带宽(10Mbps)上下波动，没有明显差别。

（2）解决 BufferBloat 问题

网络拓扑如图 6。设置 r1 的 maxqlen=1000，h1-r1 带宽为 100Mbps，变更 r1 丢包策略和 r1-h2 链路带宽，复现的测量结果如图 7。

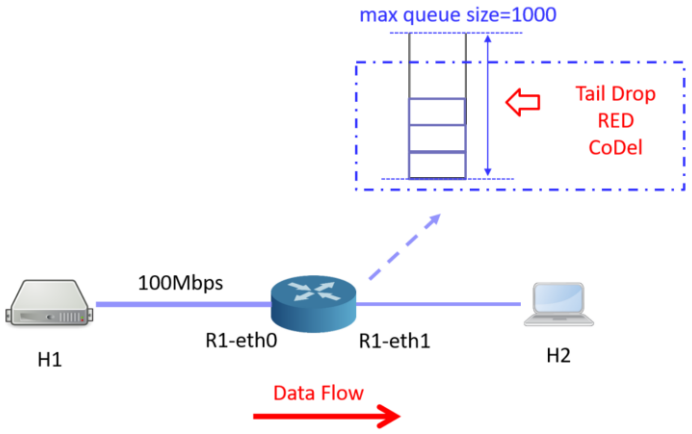

图 6

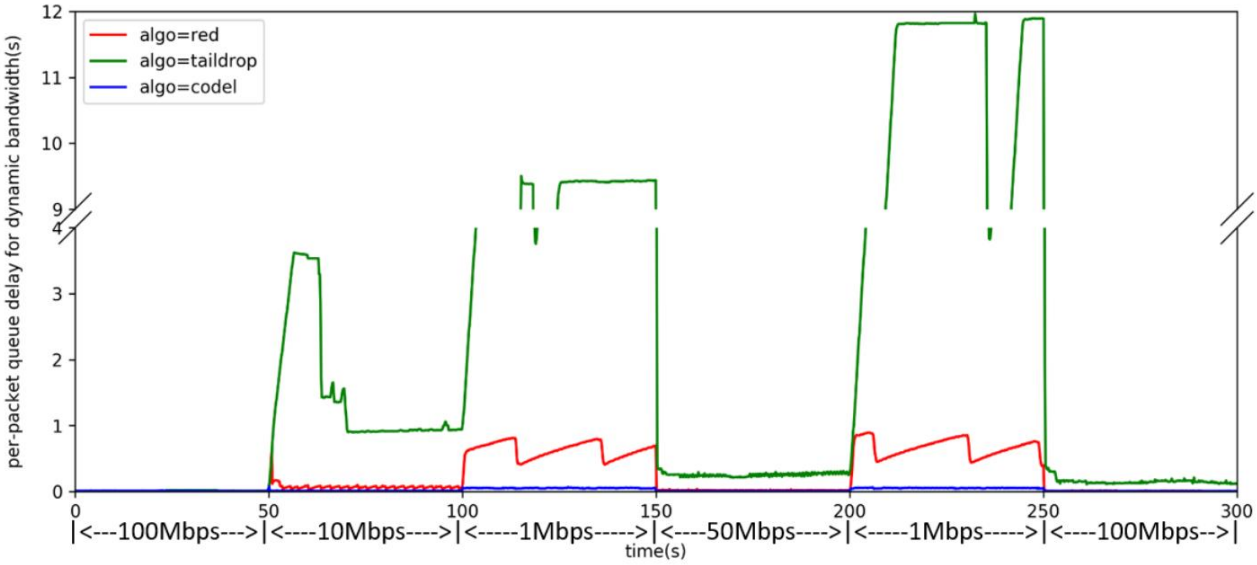

图 7 不同策略下包延迟(即 rtt)随带宽、时间变化关系(maxqlen=1000)

从图 7 可以看出，在 0-50s 和 250-300s，r1 的到达速率小于等于发送速率（即 h1-r1 带宽小于 r1-h2 带宽，前者为 100Mbps），包延迟很小，没有 bufferbloat 现象。当到达速率高于发送速率时，数据包在队列中滞留，尤其是当 r1-h2 带宽减小到 1Mbps 时（对应 100-150s 和 200-250s），bufferbloat 现象明显，taildrop 策略下包延迟达到数秒甚至十几秒，而 red 策略下包延迟接近 1s，表现较好，codel 策略下包延迟没有明显增加，保持在毫秒级，表现最好。

5、遇到的问题

（1）Ping 程序结果输出的持续时间比设定时间短很多。多次重复后仍难以解决。最后采用两种方法。首先是增加测量间隔，可以使 cwnd 和 qlen 的测量结果更大概率地正常输出。但有时 rtt 的测量结果无法输出。于是我改用 iperf 来获得 rtt 的测量值。

6、调研题

导致 Bufferbloat 问题的三个关键因素：队列长度，队列管理机制，和拥塞控制机制。同时，分别从上述三个角度都可以解决 Bufferbloat 问题。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。

（1）BBR

Bufferbloat 的原因是数据的发送带宽小于到达带宽，导致数据包在队列中的滞留。传统的拥塞控制算法(CUBIC)以丢包作为减小拥塞窗口的信号，尽管这种算法可以达到最大的传输速率，但是它是以高延迟作为代价的。因为当队列过大且开始丢包时，发送的数据超出 BDP（时延带宽积）的部分滞留在发送队列中，造成了高延迟。

BBR(bottleneck bandwidth and round-trip propagation time)通过估计 BtlBw 和 RTprop 来达成以下两个条件，从而在达到最高的吞吐量的同时保持最低时延：

速率平衡：瓶颈带宽的数据到达速率与 BtlBw 相等；

填满管道：所有的在外数据（inflight data）与 BDP（带宽与时延的乘积）相等。

TCP 记录每个报文的离开时间从而计算 RTT。BBR 必须额外记录已经发送的数据大小，使得在收到每一个 ACK 之后，计算 RTT 及发送速率的值，最后得到 RTprop 和 BtlBw 的估计值。

BBR 将它的大部分时间的在外发送数据都保持为一个 BDP 大小，并且发送速率保持在估计得 BtlBw 值，这将会最小化时延。但是这会把网络中的瓶颈链路移动到 BBR 发送方本身。所以，BBR 周期性增加发送速率和在外报文。如果 BtlBw 没有改变，说明 BBR 不是瓶颈链路，

BBR 将发送速率复原。否则，BBR 会立即更新 BtlBw 的估计值，从而增大了发送速率。通过这种机制，BBR 可以以指数速度非常快地收敛到瓶颈链路。

BBR 的拥塞控制机制有四个阶段：STARTUP,DRAIN,PROBE_BW,PROBE_RTT。

第一阶段 STARTUP 类似于 TCP 的慢启动。通过指数增加的方式增加发送速率，希望能够快速探测瓶颈带宽。当判断连续三个往返时间带宽的提升不足 25%时，表示已经达到了瓶颈带宽，状态切换至 DRAIN。

DRAIN 阶段排空 STARTUP 阶段造成的网络缓存，使飞行包的大小等于 BDP，从而进入 PROBE_RTT。

第三个阶段 PROBE_BW,BBR 的大部分时间都在该状态。当 BBR 测量到瓶颈带宽和最小 rtt，并且 inflight data 等于 BDP 后，便开始以一个稳定的匀速维护着网络状态，偶尔小幅提速探测是否有更大带宽，然后再小幅降速公平的让出部分带宽。

最后一个阶段 PROBE_RTT。在 PROBE_BW 阶段 10s 内没有更新 RTTProp 时，就会进入这个阶段。在这个阶段内 BBR 的发送窗口会被固定为 4 个包，从而排空链路上的数据包，测量真实的 RTT。接下来回到 PROBE_BW 阶段。

（2）HPCC

HPCC 是一种用于大型高速网络的新型流控机制，利用网络内遥测技术(INT)来获取精确的链路负载信息，并精确地控制流量。HPCC 可以快速利用空闲带宽，同时避免拥塞，并可以保持接近于零的网络内队列，实现超低延迟。

发送方发送的每个数据包将由接收方确认。在从发送器到接收器的数据包传播过程中，沿路径的每个交换机利用其 INT 功能来插入一些数据，这些数据报告了包括时间戳（ts），队列长度（qLen），发送字节数（tx 字节）和链路带宽容量（B）的信息。当接收方获取数据包时，它会将记录的所有元数据复制到 ACK 消息中发送回发送方。每次收到带有网络负载信息的 ACK 时，发送方调整发送窗口，使 inflight data 略小于 BDP，这样就避免了数据包在队列中的滞留，解决了 bufferbloat。