Bufferbloat 实验设计报告

中国科学院大学

页

2022 年 4 月 26 日

一、数据包队列管理实验

1. 实验内容

（1）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，执行 reproduce_bufferbloat.py，设置最大队列长度(maxqlen)为 20，40，60，80，100，得到 CWND、qlen、RTT 这 3 个时间曲线图。

（2）解决 BufferBloat 问题: 执行 mitigate_bufferbloat.py，设置 maxqlen 为 1000，测量在三种策略（red,taildrop,codel）下，rtt 随时间的变化。

（3）完成调研题。

3. 实验结果与分析

（1）重现 BufferBloat 问题

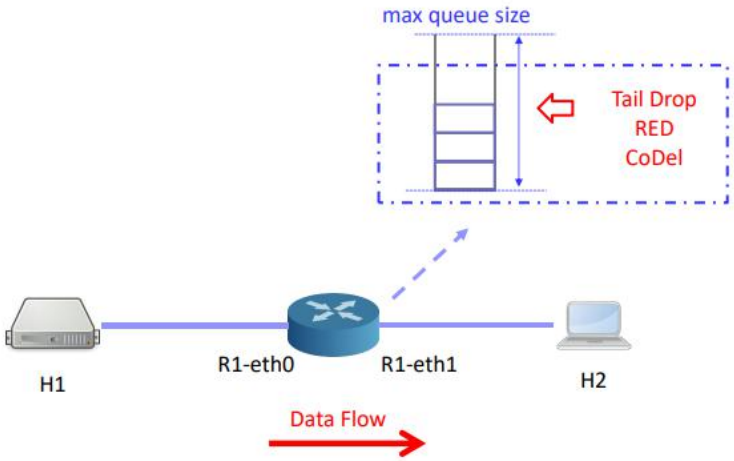

图 1

网络拓扑如图 1。变更 maxqlen，可以得到不同 maxqlen 大小下发送方 cwnd (拥塞窗口数量,)随时间变化的曲线，接收方 qlen ([实时]队列长度，下同) 随时间变化的曲线，RTT (往返时延，下同) 随时间变化的曲线和吞吐率随时间的变化曲线如下：

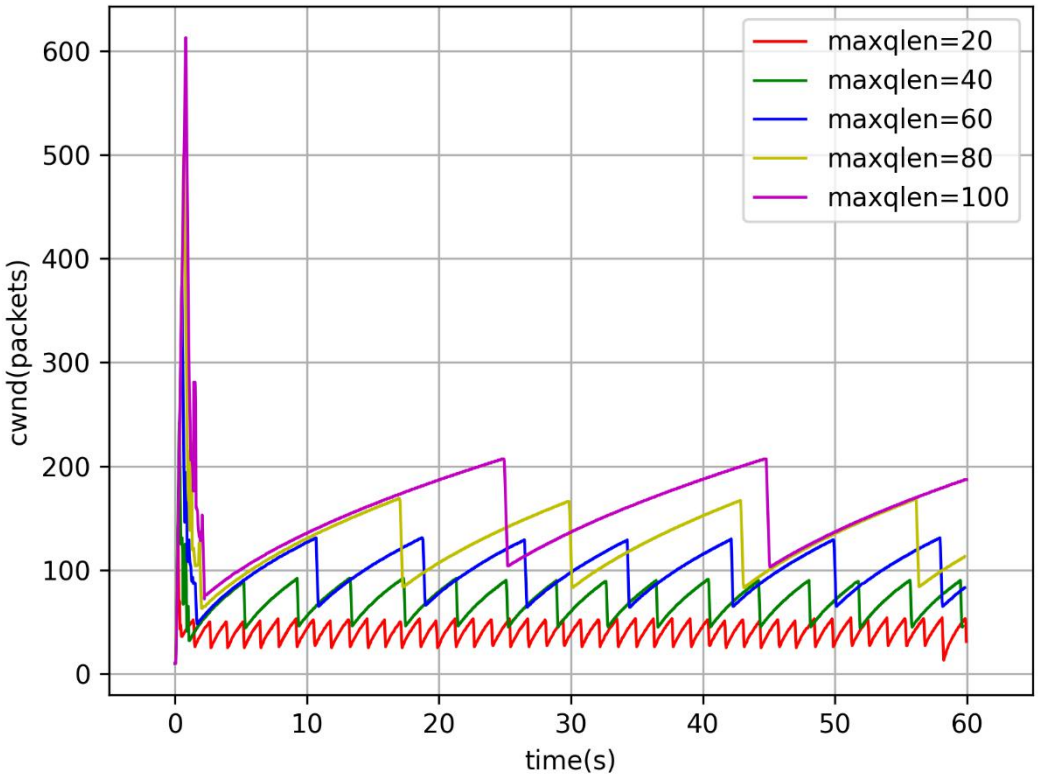

图 2 发送方拥塞窗口值(cwnd)随时间的变化

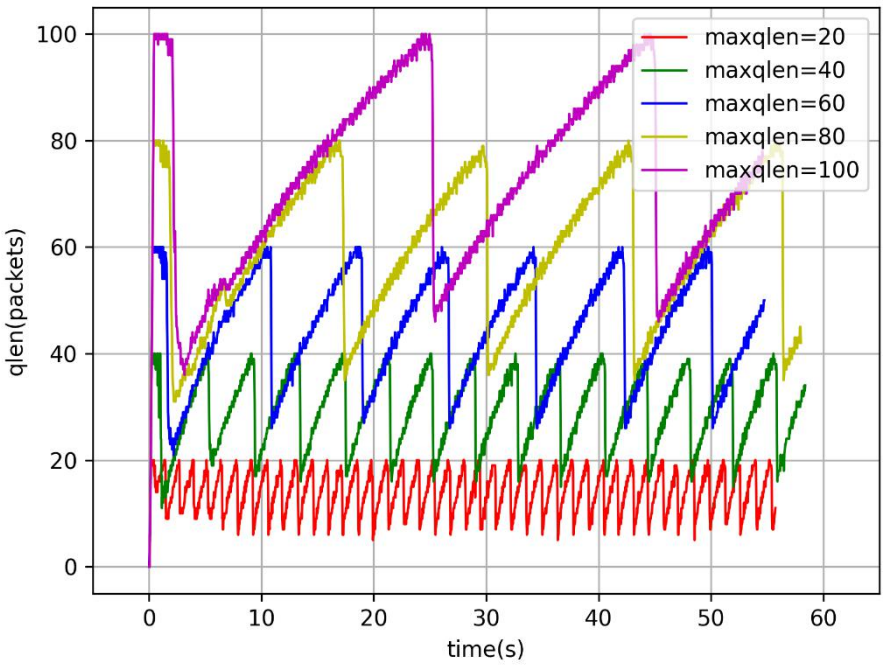

图 3 路由器队列长度(qlen)随时间的变化

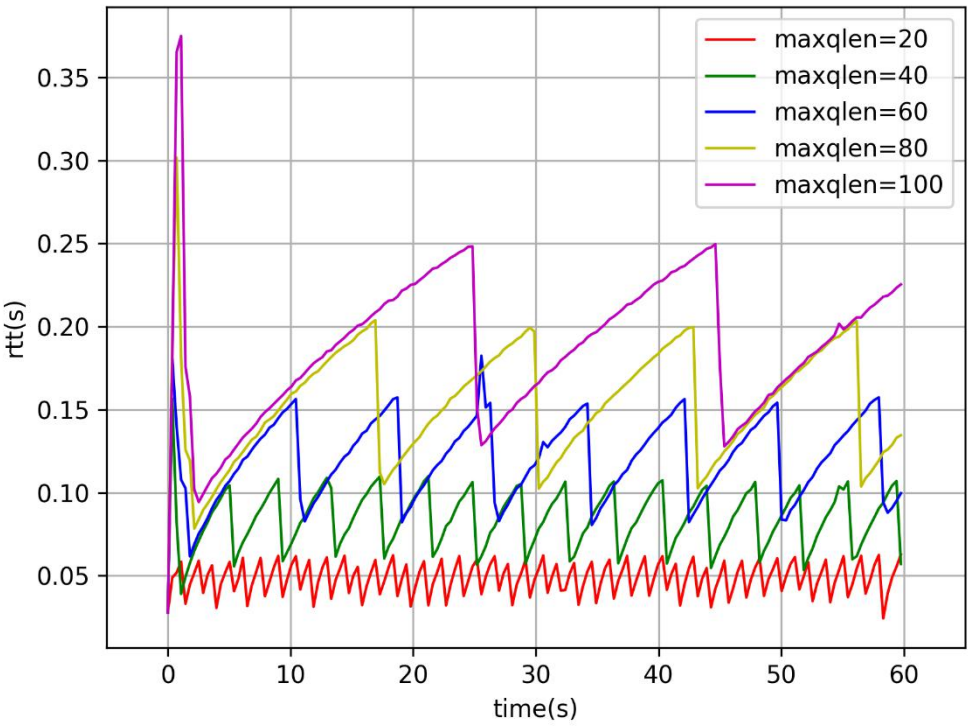

图 4：rtt 随时间的变化

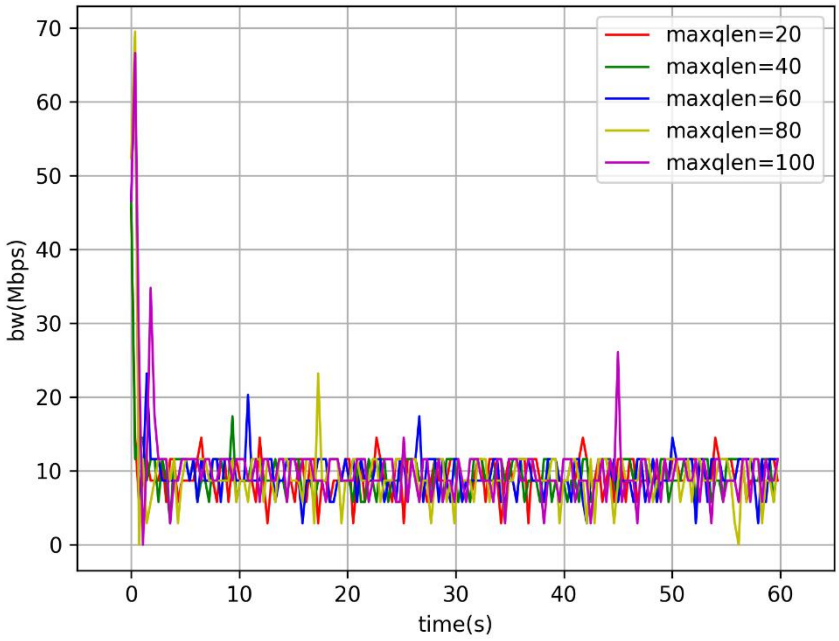

图 5：吞吐率随时间的变化

比较图 2，图 3，图 4，可以看到, 不同的 maxq (最大队列长度, 下同) 值下,它们的行为都很相似。当 maxqlen 较小时，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。从 0 开始,迅速增加，当 maxqlen 增大到一定程度时，由于瓶颈链路的存在，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，较为快速地回落到较低值。TCP 拥塞控制机制迅速减小发送窗口，使 cwnd,qlen,rtt 均迅速降低。由于 TCP 拥塞控制算法的动态调控的影响，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，，在周期中先缓慢增长后迅速下降.。最后，由图 5 可以看出，在迅速的增加回落之后，不同 maxqlen 的吞吐率均在瓶颈链路带宽(10Mbps)上下波动，同样保持在一个相对稳定的状态下。

（2）解决 BufferBloat 问题

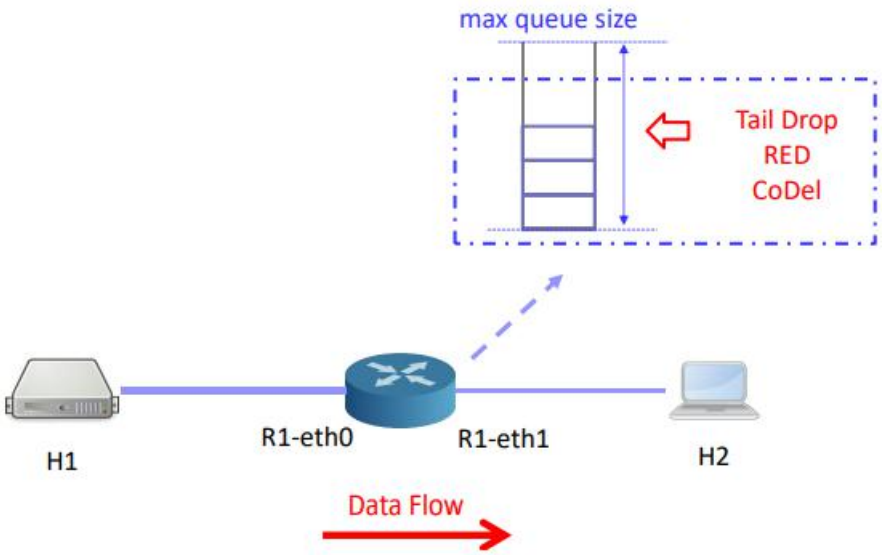

图 6

网络拓扑如图 6 所示，改变丢包策略得到结果如图 7 所示：

在解决 BufferBloat 问题上，实验中使用了三种算法： TailDrop ， RED 和 CoDel 。

- TailDrop ：尾部丢弃算法，当队列满时，直接将新到达的数据包丢弃。此算法不需要任何参数，但是因为等缓冲区满了才开始丢包，此时网络状态已经非常拥塞，所以该算法没有主动避免拥塞的功能。
- RED ：随机早期检测算法 Random Early Detection 。在此算法中，队列满之前就开始主动地概率性丢包，丢包概率与队列长度正相关。此算法可能提前处理可能出现的拥塞情况，但是由于需要设定的参数很多，调参十分不易，因此现代路由有此功能也往往不会使用
- CoDel ：控制时延算法 Controlled Delay 。 此算法会控制数据包在队列中的时间并以此为度量指标（而不是队列长度）。具体来说，当包停留时间超过 target 值时，将该数据包丢弃，并且设置下次丢包时间，直到所有包的停留时间都小于 target 值。此算法可以提前处理拥塞，而且需要的参数个数比 RED 算法少，降低了实际实现的难度。

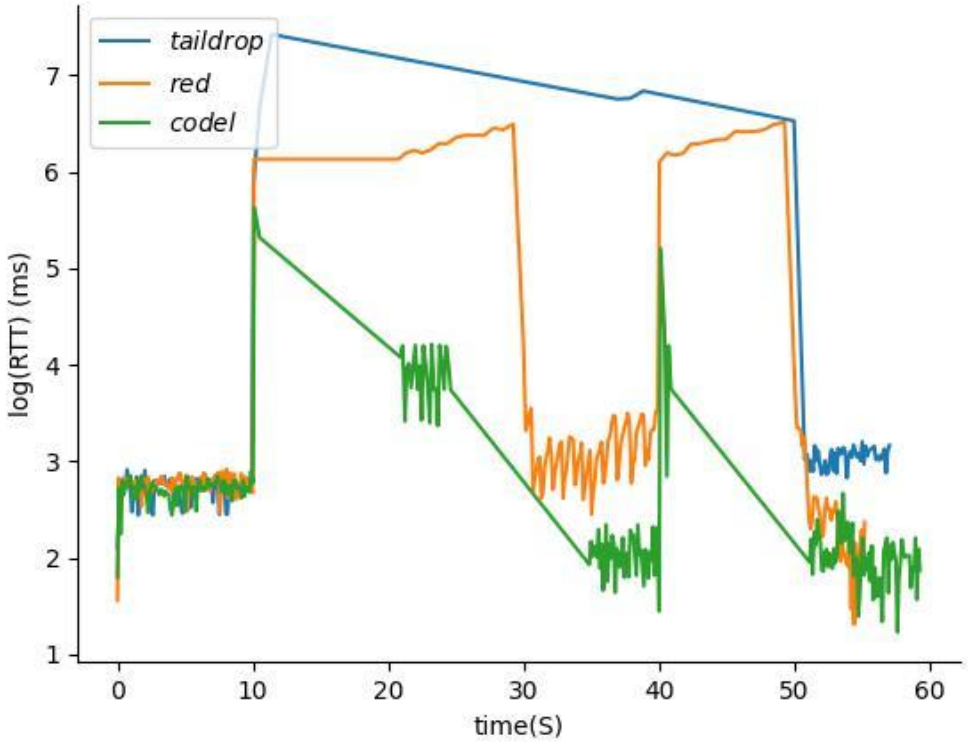

图 7：不同策略下包延迟(log(rtt))随时间变化曲线

观察图 7，一开始网络处于较为通畅的状态， rtt 维持在较低的水平，三种算法此时还没有发挥作用。当大量数据包开始堆积时，TailDrop 算法表现最差，在 10-50s 内保持在很高的水平，TailDrop 算法在面对大量涌入的数据包时将大量数据包被直接丢弃， rtt 迅速飙升。RED 算法相比 TailDrop 算法效果有一定的提升，RED 算法会根据队列长度进行丢包，让队列长度维持在较低的水平。但是观察图像我们可以发现在 10-30s 和 40-50s 时 RTT 也保持在很高的水平，高延迟时间占比同样很长。效果并不是十分理想。CoDel 算法表现最优可以将 rtt 维持在较稳定且较低的水平，因为 CoDel 算法能比 RED 算法更早让发送端意识到拥堵，所以 CoDel 算法可以更快清空信道，降低 rtt 的波动幅度。

4. 研讨题

拥塞控制机制对 Bufferbloat 的影响：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。

答：

（1） BBR

BBR 的名称实际上是 bottleneck bandwith and round-trip propagation time 的首字母缩写，表明了 BBR 的主要运行机制：通过检测带宽和 RTT 这两个指标来进行拥塞控制。 BBR 算法的主要特点有以下几个：

1. BBR 不考虑丢包，因为丢包并不一定是网络出现拥塞的标志了
2. BBR 依赖实时检测的带宽和 RTT 来决定拥塞窗口的大小：窗口大小 = 带宽 * RTT。

BBR 既然不把丢包作为拥塞出现的信号，就需要找到其他机制来检测拥塞是否出现。Vegas 算法基于时延来判断是否出现了拥塞，Westwood 算法基于带宽和 RTT 来决定拥塞窗口的大小，但是受限于 linux 拥塞控制实现的原因，Westwood 计算带宽和 RTT 的方式十分粗糙。BBR 也采用了和 Westwood 一样的方式，但是它的作者同时改进了 linux 拥塞控制的实现，使得 BBR 能够得到更完全的控制。一个网路链路能够传输的最大吞吐取决于这条网路链路上的物理时延（Round-Trip Propagation Time，在 BBR 中简写为 RTprop）与链路上速度最低的一段的带宽（Bottle-neck Bandwidth，在 BBR 中简写为 BtlBw）的乘积。这个乘积叫做 BDP（Bottle-neck Bandwidth Delay Production），即 BDP=BltBw x RTprop，也就是将链路填满数据同时不填充中间链路设备缓冲的最大数据量。BBR 追求的就是数据发送速率达到 BDP 这个最优点。

BBR 算法的各个阶段：

在连接建立的时候，BBR 也采用类似慢启动的方式逐步增加发送速率，然后根据收到的 ack 计算 BDP，当发现 BDP 不再增长时，就进入拥塞避免阶段（这个过程完全不管有没有丢包）。在慢启动的过程中，由于几乎不会填充中间设备的缓冲区，这过程中的延迟的最小值就是最初估计的最小延迟；而慢启动结束时的最大带宽就是最初的估计的最大延迟。

慢启动结束之后，为了把慢启动过程中可能填充到缓冲区中的数据排空，BBR 会进入排空阶段，这期间会降低发送速率，如果缓冲区中有数据，降低发送速率就会使延时下降（缓冲区逐渐被清空），直到延时不再下降。

排空阶段结束后，进入稳定状态，这个阶段会交替探测带宽和延迟。带宽探测阶段是一个正反馈系统：定期尝试增加发包速率，如果收到确认的速率也增加

了，就进一步增加发包速率。具体来说，以每 8 个 RTT 为周期，在第一个 RTT 中，尝试以估计带宽的 5/4 的速度发送数据，第二个 RTT 中，为了把前一个 RTT 多发出来的包排空，以估计带宽的 3/4 的速度发送数据。剩下 6 个 RTT 里，使用估计的带宽发包（估计带宽可能在前面的过程中更新）。 这个机制使得 BBR 在带宽增加时能够迅速提高发送速率，而在带宽下降时则需要一定的时间才能降低到稳定的水平。

除了带宽检测，BBR 还会进行最小延时的检测。每过 10s，如果最小 RTT 没有改变（也就是没有发现一个更低的延迟），就进入延迟探测阶段。延迟探测阶段持续的时间仅为 200 毫秒（或一个往返延迟，如果后者更大），这段时间里发送窗口固定为 4 个包，也就是几乎不发包。这段时间内测得的最小延迟作为新的延迟估计。也就是说，大约有 2%的时间 BBR 会用极低的发包速率来测量延迟

（2）HPCC

HPCC 是一种用于大型高速网络的新型流控机制，它主要致力于实现三个目标：超低延迟，高带宽，高稳定性。HPCC 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。通过处理 INT 信息，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。

HPPC 总体模型如图 8 所示，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，发送方决定如何调整其流速.

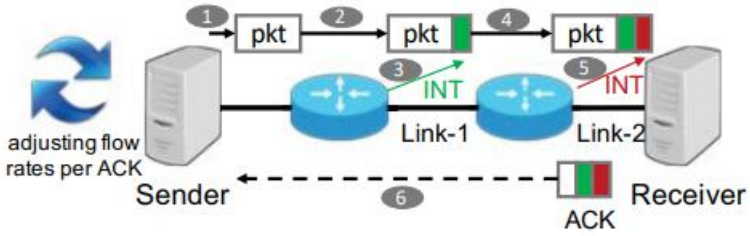

图 8

飞行包的大小直接对应了整个链路利用率。当飞行包的大小小于带宽乘 RTT，也就是带宽延时积（BDP）的时候，说明信息量不能填满这样的“管道”，则链路未被充分利用，即还没有发生拥塞，I<B×T。同理当当飞行包的大小大于带宽乘 RTT 的时候，总吞吐量将超过链路带宽，发生拥塞，即 I>=B×T。使用总的飞行包的大小来检测是否发生拥塞，并且目标是控制 I 略小于 B * T。

参考资料：

【1】 改进 TCP，阿里提出高速云网络拥塞控制协议 HPCC。

【2】 Li Y, Miao R, Liu H H, et al. HPCC: High precision congestion control[M]//Proceedings of the ACM Special Interest Group on Data Communication. 2019: 44-58.

【3】 Sigcomm2019 High Precision Congestion Control（HPCC）论文阅读笔记。