计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(4):193−198 [doi: 10.15888/j.cnki.csa.007905]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于改进 BBR 的数据报拥塞控制协议①

刘明昊

(广东工业大学 计算机学院, 广州 510006)
通讯作者: 刘明昊, E-mail: mario@mail2.gdut.edu.cn

摘　要: 数据报拥塞控制协议 (Datagram Congestion Control Protocol, DCCP) 是提供拥塞控制和不可靠传输特点的实时多媒体基础协议, DCCP 中的 CCID2 算法仍然采用 AIMD 的控制机制, 这种传统的 Loss-Base 拥塞控制模型已经不适用于目前高 BDP 的网络环境, 容易引起缓冲区膨胀现象, 导致网络延迟增加和抖动等问题. 与 Loss-Base 的算法相比, BBR 算法可以有效地控制网络延时, 最大限度避免网络排队的情况, 在丢包率较高的情况下仍可以保持一定的带宽利用率和较低的链路延时, 因此适合于 DCCP 实时流媒体的应用的协议. 本文在 DCCP 中引入了 BBR 算法并做相应的改进, 增加了丢包率检测模型, 使用延时与带宽积模型的拥塞控制算法对上述问题进行改进. 通过模拟实验证明, 本方法在高负载情况下连接的平均延迟相比 CCID2 降低了 20%, 在丢包率较高的环境下也能保持良好的吞吐量.

关键词: CCID2; 数据报拥塞控制协议; 拥塞控制; BBR; 缓冲区膨胀

引用格式: 刘明昊.基于改进 BBR 的数据报拥塞控制协议.计算机系统应用,2021,30(4):193–198. http://www.c-s-a.org.cn/1003-3254/7905.html

Datagram Congestion Control Protocol Based on Improved BBR

LIU Ming-Hao

(Department of Computer Science, Guangdong University of Technology, Guangzhou 510006, China)

Abstract: As a substrate protocol for real-time multimedia application, DCCP is featured by congestion control and unreliable transmission. However, the congestion control algorithm CCID2 in DCCP is still based on AIMD, which will cause bufferbloat, longer network delay, and jitters. Hence, such a Loss-Base model is no longer suitable for the high-BDP environment. In contrast, the BBR algorithm can effectively control the network delay, minimize the network queuing, and maintain high bandwidth utilization and low link delay even at a high packet loss rate. Therefore, it is suitable for the real-time multimedia applications with DCCP. This study adds a detection model for packet loss rates to the BBR algorithm after its introduction to DCCP and applies the congestion control algorithm in the model of delay and bandwidth product to addressing the above-mentioned problems. Compared with CCID2, the proposed algorithm reduces the average delay by 20% under heavy loads and can produce a large throughput at a high packet loss rate.

Key words: CCID2; Datagram Congestion Control Protocol (DCCP); congestion control; BBR; bufferbloat

DCCP 作为一种适用于实时多媒体流的传输协议, 拥有 UDP 不可靠传输特性和 TCP 的拥塞控制机制的特点, 它作为一种通用的底层协议, 避免了在应用层上进行拥塞控制算法设计与实现. 围绕 DCCP 的研究包括了提高多媒体流传输的实时性, 优化传输速率稳定性等方面. 关键的优化点围绕 DCCP 中的 AIMD 机制展开, 文献 [1] 基于卡尔曼滤波对 CCID2 算法[2] 进行改进, 但这种算法的复杂度可能给网络资源带来过多

① 收稿时间: 2020-08-12; 修改时间: 2020-09-15, 2020-10-06; 采用时间: 2020-10-21; csa 在线出版时间: 2021-03-30

的消耗. 文献 [3] 考虑了误码丢包, 使用信道繁忙比来检测网络拥塞的方法对 CCID2 进行优化, 适用于特殊的 Ad hoc 网络, 但是这种优化并没有改变算法 AIMD 的本质, 无法使算法在获得高吞吐量的同时保持低延迟. 由于丢包作为这类算法的唯一输入信号, 算法的正常运行对链路的丢包率有一定的要求, 在丢包率较高的长肥管道环境下, 其发送窗口会收敛到很小的值[4]. 随着网络中间设备的队列深度扩增, 基于丢包的拥塞控制算法已经不再适合如今的网络环境, 这类算法的拥塞避免阶段会逐渐加大发送窗口直至填满瓶颈队列, 正是这种机制本身导致了链路拥塞, 造成了网络延时的波动. 在链路瓶颈处保持最大带宽和最小延时的状态是拥塞控制的目标, 但这个状态曾被证明不可能由分布式算法收敛[5]. 最近谷歌提出一种基于延时带宽积的算法 BBR (Bottleneck Bandwidth and RTT)[6], 使用了交替测试链路的最大带宽与最小的 RTT 的方法, 通过估计链路 *BDP* 的思路解决了这个问题. BBR 算法致力于收敛到最佳的拥塞控制点, 因此适合 DCCP 这种对于延时和带宽敏感的流媒体传输协议, 本文在 DCCP 中引入 BBR 算法, 考虑到 DCCP 与 TCP 传输模型的差异, 对链路瓶颈带宽的计算方法进行适配, 为了增加算法抵抗丢包的能力, 此外, 本文在 BBR 的基础上加入动态测量丢包率的机制, 优化 BBR在固定增益系数下存在的失速问题. 经过仿真实验验证, 改进后的算法可以在高吞吐量的状态下保持低延时, 提高了在高丢包率的网络环境下连接的吞吐量.

1 CCID2 算法模型

CCID2 是一种类 TCP 的拥塞控制算法, 并不保证数据可靠性, 它使用 AIMD 机制计算拥塞窗口限制发送速率以适应不同的网络环境. CCID2 应用在 DCCP 上, 其拥塞窗口的单位与 TCP 不同, 在 TCP 上单位为字节, 而在 DCCP 上单位为数据报数量, 传输的单位为数据包. CCID2 使用类似 TCP-SACK 方法来实现拥塞控制机制, 然而, 作为一种 Loss-Base 的拥塞控制算法, 它存在以下缺点:

1) 带宽利用率低, CCID2 本质上属于丢包事件驱动的拥塞控制算法, 当链路的丢包率上升至 1% 以上的时, 算法基本上无法正常工作.

2) 缓冲区膨胀问题 (bufferbloat)[7,8], CCID2 算法在拥塞避免阶段会线性探测链路最大带宽, 逐步增加其发包速率直至队列满载, 如图 1 的 (1) 阶段. 随后由于队列满导致的丢包事件会反馈回发送端, 发送端降低一半的发送窗口, 希望排空队列的数据包, 如图 1 的 (2) 阶段. 由于目前网络中间设备的队列大小都已经大幅提高, 这种拥塞控制模型已经很难适应现在的网络环境, 减半窗口的做法不一定可以排空队列的数据包, 实际上并没有完全缓解网络的拥塞压力. 算法在减窗后马上又开始进行 (1) 阶段, 导致了链路上一直处于高负载的状态, 不仅造成了链路 RTT 增加而且限制了基于延时的算法的使用[9], 如 VEGAS[10] 和 FAST[11]. 实际上, 当网络中间设备当队列开始积累数据包当时候, 网络当拥塞就已经发生, 由于丢包作为该拥塞控制的唯一反馈信号, 这种模型无法获取除了丢包外的任何信息, 对拥塞状态的判断存在延后的情况. 为了解决以上问题, 常用的做法是在中间设备上引入主动队列管理 AQM[12], 使用简单的 RED[13] 算法, 根据路由缓冲队列的负载程度进行随机丢包, 作为给予连接降窗的信号, 文献 [14] 中指出这种做法在中间设备引入了 AQM 配置复杂度, 增加了维护成本. 总体上, 缓冲区膨胀并不是队列本身的问题, 而是由终端拥塞控制算法导致的, 在终端进行优化是首要的选择.

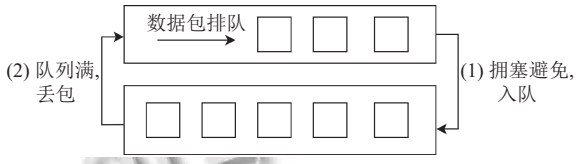

图 1 Loss-Base 拥塞控制算法事件循环

2 BBR 算法模型

BBR 算法与基于丢包的拥塞控制算法不同, 在 BBR 算法的模型中, 定义了链路的最大负载为链路的时延 RTT 与链路带宽的乘积 (Bandwidth Delay Product, *BDP*), 如果发送端往网络注入数据包的速度超过了 *BDP*, 中间设备也无法转发更多的数据包, 后续加入数据包会开始排队导致链路延时开始增加. 在 BBR 中链路的 RTT 由 (1) 式计算:

$$RTProp' = RTProp + \min(\eta_t) \tag{1}$$

其中, *RTProp* 的值为链路延迟, 由链路物理长度决定, 而η_t值为 ACK 延时机制, 数据包协议栈处理所消耗的时间, 两者之和为算法的测量值, $RTProp'$的下限为链路物理时延. 对于带宽的计算, BBR 采用了 *Delivery-*

Rate 的最大值作为瓶颈带宽 *BTLBW*, 该值的上限为链路的物理带宽, 通过两者乘积可以估计出链路的 *BDP*, 从而控制发送的速率, 避免对中间设备造成排队的压力.

BBR 算法模型如图 2 所示.

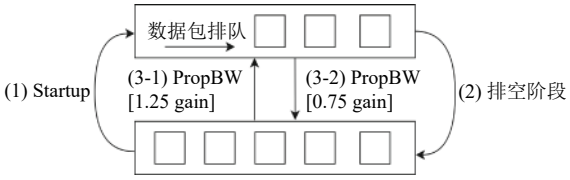

图 2 BBR 算法事件循环

算法在 (1) 阶段会探测链路的最大带宽, 该阶段链路会出现排队的情况, 随后在 (2) 阶段按同样的比例排空发送过多的数据包, 随后进入稳定的 (3) 阶段, *PROPBW* 会周期性探测链路的带宽. 在 (3-1) 的探测阶段内, 算法会少量增加发送数量以探测链路空闲带宽, 随后在 (3-2) 阶段会按相应的比例降低发送数量, BBR 避免了 Loss-Base 算法一直填满缓冲队列的做法, 因此可以同时保持高带宽利用率和较低的延迟. 实际上, 链路最低延时与最大带宽无法同时测量, 要测量带宽必须往链路发送过量的数据以计算获得最大的实际带宽, 此时由于大量数据包排队, 必然会导致延时的增加. 另一方面, 要测量链路延时, 必须保证连接本身没有对中间设备进行排队, 这时需要减少数据的发送量. BBR 使用交替测试 RTT 与链路带宽的做法解决了上述问题. 尽管如此, BBR 仍然存在以下的缺陷:

1) BBR 算法在 STARTUP 过程中, 为了适应不同链路的带宽情况, 使用二分搜索的方法探测链路的 *BDP*, 具体计算过程如下 (下面使用 *G* 代替 *PACING_GAIN*):

① 定义 STARTUP 过程的发送速率 *SendingRate* 为 RTT 时间的函数 (其中 α 为常系数):

$$SendingRate(t) = \alpha 2^t \tag{2}$$

② 为得到 *PACING_GAIN*, 令当前时间为 t–2, 对于 [t–2, t–1] 区间有:

$$SendingRate(t-2) = \alpha 2^{t-2} \tag{3}$$

③ 由于 *PACING_GAIN* 与当前 *SendingRate* 的积定义了下一个周期的 *SendingRate*, 从而有:

$$G \times SendingRate(t-2) = BDP_{[t-1,t]} \tag{4}$$

④ 对 RTT 归一化处理后, 计算出增益系数的值:

$$\begin{cases} G \times SendingRate(t-2) = BDP_{[t-1,t]} \\ G = \dfrac{BDP_{[t-1,t]}}{SendingRate(t-2)} \\ G = \dfrac{\alpha \int_{t-1}^{t} 2^t dt}{\alpha 2^{t-2}} \\ G = \dfrac{2}{\ln 2} \approx 2.89 \end{cases} \tag{5}$$

使用该系数可以使探测算法在$\log_2 BDP$个 RTT 内收敛到瓶颈带宽 *BTLBW*, 但会对瓶颈链路缓冲区造成过多压力[15], 考虑到实时应用容易受链路带宽变化和缓冲区膨胀问题的影响, 因此可以适当降低 STARTUP 的 *PACING_GAIN*.

2) 文献 [16] 指出, BBR 算法在丢包率过高的环境下并不能一直保持有效的吞吐量, 丢包率主要由 *PACING_GAIN* 的正系数保证, 在丢包率波动很大的环境下, 若网络的丢包率超过固定的增益参数可以调整的范围, 会导致发送速率逐步收敛到低值.

3 DCCP-BBR 算法模型与实现

DCCP-BBR 算法引入了带宽, 丢包率和动态增益系数计算的模型.

3.1 带宽计算

由于 CCID2 只记录数据包序号, 没有记录数据包的大小, 为了测量链路的实际带宽, 需要记录发送窗口内的数据包序号及其对应的包大小的状态:

$$Map[seq \rightarrow packet\ size]$$

对于 ACK 的计算, 沿用原有的 ACK-Ratio 机制, 默认值为 2, 此外, 算法使用两个事件更新的带宽信息:

1) onSend() 事件

该事件发生在每一个数据包进行发送的时刻, 需要记录数据包的元信息用于后续的带宽估计:

① 记录发送的时间:

$$packet\_send\_time[seq]$$

② 记录发送时刻的送达数据量 *delivered_size* 值:

$$packet\_send\_delivered[seq]$$

③ 记录发送数据包的大小:

$$packet\_size[seq]$$

2) onAck() 事件:

该事件在每收到一个 ACK 时触发, 需要计算带宽与延时的测量值, 计算伪代码如图 3 所示.

```
onAck(packet) {
    packet.foreach(ack in ack vector) {
        delivered_size += packet_size[ack];
        pre_delivered = packet_send_delivered[ack];
        delivered = delivered_size - pre_delivered;

        packet_send_time = packet_send_time[ack];
        rtt = now - packet_send_time;

        bw = delivered / rtt;
    }
}
```

图 3 onAck 事件伪代码示意图

具体的计算流程如下:

① 累计当前数据包的大小至 *delivered_size* 中, 该变量记录连接收到 ACK 确认的总量.

② 计算数据包发送到接受完毕时候所传输的数据量 *delivered*.

③ 计算数据包送达的时间 RTT, 对于接收到多个 ACK 的 *Vector*, 取 ACK 包发送时间的最小送达所完成的时间 RTT.

④ 计算实时带宽的估计值 *bw*.

至此, 结合 *bw* 值与 *PACING_GAIN* 增益系数的乘积即可导出发送窗口的值.

3.2 丢包率估计

连接的丢包率可以在发送端也可以在接收端计算. 考虑延迟 ACK 的影响, 在发送计算时如果出现接收端 ACK 丢失的问题, 会导致计算的 *Loss-Rate* 偏大. 为了更准确地估计链路的丢包率, 本实现选择在接收端进行丢包率计算, 然后反馈在 ACK 报文中. 计算流程如下:

1) 协议保证每一个数据报序号总是单调递增的.

2) 对于每一个接收到的数据包, 判断数据包是否在 10 s 有效窗口内, 记录在窗口时间内接收到的数据总量 *W_ALL*, 丢包率在全局窗口内计算.

3) 若出现乱序到达的情况, 按照规则 2) 对丢包率进行补偿. 在每一个窗口内, 记录实际收包总量 *P_NUMS*. 最终的丢包率由式 (6) 计算:

$$Loss-Rate=\frac{P\_LOSS}{W\_ALL}=1-\frac{P\_NUMS}{W\_ALL} \tag{6}$$

3.3 增益系数计算

BBR 算法在绝大多数时间内处于 ProbeBW 状态, 其主要工作为探测链路是否有未利用的带宽资源, 算法引入了使用了 8 个阶段的 Gain Cycle 状态, 分别为 1.25, 0.75, 1, 1, 1, 1, 1, 1. 增益系数在上述数组中循环取值, 算法在每一个阶段持续时间约为 *RTprop*. BBR 首先使用 1.25 系数增加发包数量, 如果实际反馈计算的带宽增加, 意味着 BBR 可以占据链路空余的带宽. 但由于 1.25 增益阶段可能导致的链路瓶颈队列排队, 算法设置了 0.75 增益阶段用于排空瓶颈队列. 随后, 算法的平稳阶段采用 1 作为增益参数, 按照反馈带宽进行发包, 以此保证连接的公平性. BBR 增益系数轮转机制的参数确定主要源于以下几个方面的考虑:

1) 平稳阶段的长度决定了 BBR 探测带宽的间隔, 短的间隔提升了探测过程的抢占速度, 长的间隔用于保证公平性, 两者并不能同时兼顾.

2) 较大的增益系数可以增加算法竞争性, 在链路有空闲资源的时候可以更快收敛到新的 *BDP*, 但也会对链路延迟产生较大的波动.

3) 较小的增益系数一定程度上减少了竞争性, 同时也意味着每一个 *ProbeBW* 阶段的收益会降低, 收敛到新 *BDP* 需要更长的时间.

如果增益系数过小, 正增益的效果会完全被丢包副作用抵消, 从而导致发送速率持续下降. 由于 BBR 算法增益系数是固定的, 1.25 增益系数只能保证在 20% 丢包率以下工作, DCCP-BBR 采用动态计算增益系数的方法. 引入丢包率 (*Loss-Rate*) 参数, 使用丢包率导出实时的增益系数, 对于每一个反馈的 *Loss-Rate*, 发送端在每一个新的 *ProbeBW* 周期, 需要调整 PACING_GAIN 系数. 具体调整公式如下:

对于估算的 *BTLBW* 值, 其增益系数需要满足式 (7) (下面使用 *BW* 代替 *BTLBW*):

$$PACING\_GAIN\times BW\geq BW \tag{7}$$

上式保证正增益后的值大于当前的瓶颈带宽, 否则最大 *BW* 的估计值会不断地出现负反馈的情况, 引入丢包率后, 需要保证:

$$PACING\_GAIN\times BW\times(1-LossRate)\geq BW$$
$$PACING\_GAIN\geq\frac{1}{(1-LossRate)} \tag{8}$$

考虑到 *PACING_GAIN* 过大会对链路造成压力, 取 1.5 为上限值, 最终的系数计算如下 (其中 *LR* 为丢包率):

$$PACING\_GAIN=\begin{cases}1.25 & 0.75 & 1 & 1\cdots & LR<=0.2\\ 1.50 & 0.50 & 1 & 1\cdots & LR>0.3\\ \dfrac{1}{1-LR} & \dfrac{1-2\times LR}{1-LR} & 1 & 1\cdots & 0.2<LR<=0.3\end{cases} \tag{9}$$

3.4 算法复杂度分析

算法使用了 Hashmap 跟踪发送窗口数据包状态, 对于发送方, RTT 与 *BW* 的计算需要获取对应数据包的发送的时间 *packet_send_time*, 发送时刻的已送达数据量 *delivered_size* 与数据包的大小 *packet_size*, 三者可共享一个哈希表, 每一条记录需要约 20 字节, 计算过程可在 O(1) 时间完成. 哈希表所需空间为发送窗口/数据包大小 (发送窗口上限为链路 *BDP*). 对数据包按 *MTU* 大小估计, 记录所需的空间约为:

$$20\times\frac{BDP}{MTU} \tag{10}$$

由于接收方不需要维护数据包的元信息 (*meta data*), 只需要保留 10 s 窗口的历史记录, 如图 4 所示.

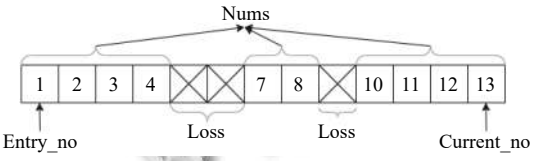

图 4 丢包率计算示意图

窗口内需要标记入口序列号 *entry_no*, 当前序列号 *current_no* 与窗口内接收到数据包的数量 *nums*. 并根据下式导出窗口内的丢包率:

$$LossRate = 1-\frac{nums}{current\_no-entry\_no} \tag{11}$$

丢包率反馈过程计算复杂度为 O(1), 不需要额外空间.

4 仿真模拟实验

为了验证基于 BBR 改进后的 DCCP 协议的性能, 本实验使用网络模拟器 MININET[17] 模拟网络环境. 在两台主机上面进行带宽测试, 在不同丢包率环境下 (0~0.4), 分别与 CCID2, 原 BBR 算法的实现进行对比. 测试环境的网络拓扑如图 5 所示.

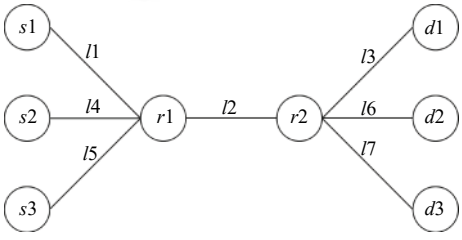

图 5 仿真实验网络拓扑

图 5 中路由器 *r*1 与 *r*2 之间为测试的瓶颈链路, *s*1–*s*3 为发送端, *d*1–*d*3 为接收端, 所有发送端与 *r*1 连接, 接收端与 *r*2 连接. 瓶颈链路的带宽设置为 100 Mb/s, 其余所有链路的带宽均为 1 GB/s, 路由器与客户端连接不附加往返延迟, 路由器 *r*1 与 *r*2 之间 *l*2 的往返延迟设置为 100 ms, 路由队列长度 *max_queue_size* 设置为 4 k, 队列控制算法使用 Drop-Tail.

4.1 DCCP-BBR 与 CCID2 负载时链路延时测试

为了给链路加上负载, 使用 *s*2–*s*3 两个发送端分别向 *d*2–*d*3 建立 TCP 连接, 并且使用 Vegas 算法进行不限速数据发送, 目的是将路由器 *r*1–*r*2 之间的负载填充至瓶颈值, 但保持路由器缓冲区的空闲. 随后在 *s*1 分别使用 DCCP-BBR, DCCP-CCID2 与 *d*1 进行链路延时的测试.

图 6 描述了两种算法在传输时间为 15 s 窗口内的网络延时, 显示了从连接启动到平稳状态时候的性能. 其中使用 CCID2 的连接进入后迅速增大了链路的延迟, 进入平稳状态后达到高吞吐量后无法维持低 *RTT*, 这是因为 CCID2 会一直耗尽链路的缓冲队列直到出现 Drop-Tail 丢包. BBR-DCCP 则获得比较均衡的结果, 表现出较低的侵略性, 仅在 STARTUP 阶段探测链路最大带宽的时候主动往缓冲队列排队, 导致链路的延时暂时的上升, 当其检测到增大发送速率不再增加有效带宽的时候即退出启动阶段, 随后进入的平稳阶段会周期性地探测带宽, 导致链路延时小幅度波动, 总体上, 在保证高吞吐量的同时仍可以维持低的链路 *RTT*, 平均链路延时相比 CCID2 降低了 20%.

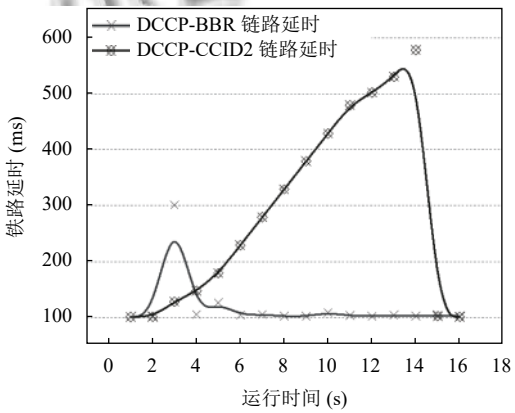

图 6 延时对比测试结果

4.2 DCCP-BBR 与 BBR 在不同丢包率下的吞吐量测试

本单元测试单连接的吞吐量, 使用同样的网络拓扑结构, 使用 *s*1 发送端与 *d*1 进行连接, 剩余节点均为关闭状态, 其他链路参数保持不变. 对 *l*2 链路设定不同

的丢包率分别进行测试.

图 7 描述了 3 种算法独立在网络中进行数据传输时, 吞吐量随丢包率变化的表现. 横坐标为链路丢包率, 由 0 逐渐增大至 0.4. 纵坐标为连接在对应丢包率下的平均吞吐量. 对于 CCID2 算法, 在丢包率高于 1% 的时候就处于不可用状态, 原因为丢包对算法的窗口计算造成持续的负反馈, 每一次丢包窗口将减少一半, 一直收敛到接近于 0. 另一方面, BBR 在丢包率高于 20% 的时候其增益系数无法平衡掉高丢包率带来的副作用, 出现吞吐量急剧下降. 改进后的 BBR-DCCP 由于其增益参数由实时丢包率计算导出, 动态调整为与链路拥塞状态匹配的值, 避免了持续的负系数反馈. 与 BBR 相比, BBR-DCCP 抵抗丢包的能力提高了 10%.

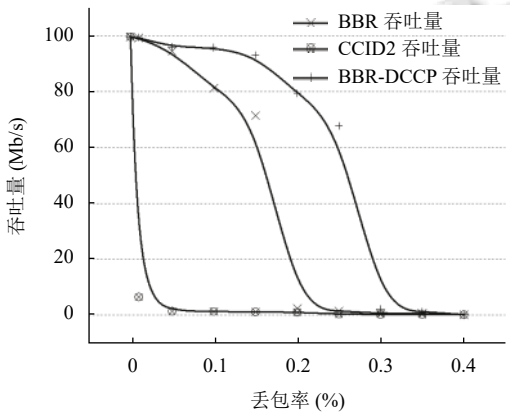

图 7 不同丢包率下吞吐量测试结果

5 结论

针对传统拥塞控制算法 CCID2 存在缓冲区膨胀的问题, 本文提出了基于 BBR 改进的数据报拥塞控制协议算法, 在原有 BBR 算法基础上增加了丢包率估计模型, 解决了在高丢包率的网络环境下连接失速的问题. 实验结果表明, 本算法可以在高带宽利用率的同时保持相对较低的延迟, 此外, 通过丢包率模型动态地调整增益系数, 算法在丢包率较高的环境下也能保持良好的吞吐量.

参考文献

1 黄玉清. 基于 DCCP 的多媒体流实时控制算法研究 [硕士学位论文]. 武汉: 华中师范大学, 2017.
2 Floyd S, Kohler E. Profile for DCCP Congestion Control ID 2: TCP-like Congestion Control. 2002.
3 卢先领, 施利利. WSN 中一种改进的 DCCP 拥塞控制机制. 计算机应用与软件, 2015, 32(6): 136–139. [doi: 10.3969/j.issn.1000-386x.2015.06.033]
4 Polese M, Chiariotti F, Bonetto E, *et al*. A survey on recent advances in transport layer protocols. IEEE Communications Surveys & Tutorials, 2019, 21(4): 3584–3608.
5 Jaffe J. Flow control power is nondecentralizable. IEEE Transactions on Communications, 1981, 29(9): 1301–1306. [doi: 10.1109/TCOM.1981.1095152]
6 Cardwell N, Cheng YC, Gunn CS, *et al*. BBR: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. Queue, 2016, 14(5): 20–53. [doi: 10.1145/3012426.3022184]
7 Cerf VG. Bufferbloat and other Internet challenges. IEEE Internet Computing, 2014, 18(5): 80. [doi: 10.1109/MIC.2014.89]
8 Gettys J, Nichols K. Bufferbloat: Dark buffers in the Internet. Communications of the ACM, 2012, 55(1): 57–65. [doi: 10.1145/2063176.2063196]
9 Chowdhury T, Alam M. Performance evaluation of TCP Vegas over TCP Reno and TCP NewReno over TCP Reno. JOIV: International Journal on Informatics Visualization, 2019, 3(3): 275–282.
10 Guo YJ, Yang XP, Wang R, *et al*. TCP adaptive Vegas: Improving of TCP vegas algorithm. Proceedings of 2014 International Conference on Information Science, Electronics and Electrical Engineering. Sapporo, Japan. 2014.126–130.
11 Jin C, Wei DX, Low SH. FAST TCP: Motivation, architecture, algorithms, performance. Proceedings of IEEE INFOCOM 2004. Hong Kong, China. 2004. 2490–2501.
12 Adams R. Active queue management: A survey. IEEE Communications Surveys & Tutorials, 2013, 15(3): 1425–1476.
13 Ismail AH, Elsagheer Z, Morsi IZ. Survey on random early detection mechanism and its variants. IOSR Journal of Computer Engineering, 2012, 2(6): 20–24.
14 Sivov A, Sokolov VA. The bufferbloat problem and TCP: Fighting with congestion and latency. Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering. 2012.
15 Hurtig P, Haile H, Grinnemo KJ, *et al*. Impact of TCP BBR on CUBIC traffic: A mixed workload evaluation. Proceedings of the 2018 30th International Teletraffic Congress. Vienna, Austria. 2018. 218–226.
16 Cao Y, Jain A, Sharma K, *et al*. When to use and when not to use BBR: An empirical analysis and evaluation study. Proceedings of Internet Measurement Conference. Amsterdam, the Netherlands. 2019. 130–136.
17 Xiang Z, Seeling P. Mininet: An Instant Virtual Network on Your Computer. In: Fitzek FHP, Granelli F, Seeling P, eds. Computing in Communication Networks. New York: Academic Press, 2020. 219–230. [doi: 10.1016/B978-0-12-820488-7.00025-6]