计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201200266

改善 STARTUP 阶段空窗现象的 BBR 单边适应算法

马力文　周　颖
南京邮电大学自动化学院、人工智能学院　南京 210023
(1319055606@njupt.edu.cn)

摘　要　为了解决校园网中应用的 BBR(Bottleneck Bandwidth and Round-Trip Time)拥塞控制算法在 STARTUP 阶段由于未收到 ACK(Acknowledge Character)而引起的时延振荡和空窗问题,提出了 BBR 单边适应算法。该算法只运行在发送端,不受网络协议和上层应用的限制。通过改善时延估计器的加权系数,设计时延瞬时平均偏差估计器,将估算结果作为时延估计器的振荡平滑因子,提高时延估计器应对时延剧烈抖动的能力。为了尽可能解决不可避免的空窗问题和序号回绕,在发送端设计了流量状态机和 STARTUP 状态机来维持较高的链路吞吐量。根据具体的传输情况将流量分为 6 种状态:new,blocked,waiting,time_waiting,running,terminated,并根据流量反馈将 STARTUP 阶段的传输性能分为 3 个状态:GOOD,NORMAL,BAD。实验结果表明,改进后的 BBR 比原有 BBR 算法在 STARTUP 阶段具有更好的传输性能,且优于目前主要应用的被动拥塞控制算法(Reno,CUBIC)。

关键词:瞬时平均偏差;时延估计器;振荡平滑因子;流量状态机;STARTUP 状态机
中图法分类号　TP393

BBR Unilateral Adaptation Algorithm for Improving Empty Window Phenomenon in STARTUP Phase

MA Li-wen and ZHOU Ying
College of Automation & College of Artificial Intelligence,Nanjing University of Posts and Telecommunications,Nanjing 210023,China

Abstract　In order to solve the problem of delay oscillation and empty window caused by the bottleneck bandwidth and round-trip time(BBR) congestion control algorithm in the STARTUP phase due to not receiving the acknowledge character(ACK) in the campus network,the BBR unilateral adaptation algorithm is proposed. The algorithm only runs on the sender,and it is not restricted by network protocols and upper-layer applications. By improving the weighting coefficient of the delay estimator,we design the instantaneous average deviation estimator of the delay and use the estimation result as the oscillation smoothing factor of the delay estimator to improve the ability of the delay estimator to deal with severe delay jitter. To solve the inevitable empty window problem and sequence number wraparound as much as possible,a flow state machine and a STARTUP state machine are designed at the sending end to maintain a high link throughput. According to the specific transmission situation,the traffic is divided into 6 states:new,blocked,waiting,time_waiting,running,terminated,and according to the traffic feedback,the transmission performance of the STARTUP stage is divided into 3 states:GOOD,NORMAL,BAD. Experimental results show that the improved BBR has better transmission performance in the STARTUP phase than the original BBR algorithm and is better than the passive congestion control algorithm (Reno,CUBIC) currently.

Keywords　Instantaneous average deviation,Time delay estimator,Oscillation smoothing factor,Flow state machine,STARTUP state machine

1　引言

思科公司预测未来 5 年全球数据中心的服务流量会以 30%的速度增长,单点数据中心已经进入高速发展期[1]。校园共享数据中心作为典型的单点数据中心,其无线连接点具有传输不稳定、单个节点处理能力弱、缓存队列浅的局限性[2]。为了进一步提高网络的传输性能,Google 于 2016 年提出了主动性拥塞算法 BBR[3],该算法通过测量代表路径的两个参数(瓶颈带宽和往返时延)来创建拥塞控制。相比被动拥塞控制算法(Reno,CUBIC),BBR 可通过周期性地减少增益系数来避免无解的缓冲膨胀[4-5]。文献[6]通过实验证明,在 WI-FI 网络中 BBR 的传输性能最好。

到稿日期:2020-12-31　返修日期:2021-06-04　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(62073172)
This work was supported by the National Natural Science Foundation of China(62073172).
通信作者:周颖(zhouying@njupt.edu.cn)

尽管文献[7]认为 BBR 在 5G 网络中对往返时延和瓶颈带宽的估计几乎是准确的，但大量实验证明，在连接开始时，BBR 对可用窗口的低估会导致链路吞吐量非常低[8-9]。校园共享数据中心提供的服务大多通过老鼠流(时延小、流量小)实现，起始阶段的性能对整体传输的影响远超 BBR 偏爱的大象流(时延长、流量大)。

针对这个问题，文献[9]使用一种客户端 TCP 拥塞控制检查工具来筛选低性能的 BBR 流。文献[10]设计了 Homa 算法，定期覆盖端到端探测，解决了流量更新达不到全局最佳收敛点的问题。文献[11]利用网络内遥测(INT)获取精准的链路信息来控制流量。文献[12]启用 VFV 的多播请求，通过优化基本路由问题来获得更大的吞吐量。文献[13]开发了 Rein，将多种拥塞控制算法分配在一台服务器上，实现了性能互补。文献[14]提出了 Corundum 算法，实现了高性能数据路径和可扩展队列的管理，可以对包传输进行准确控制。但这些设计都没有针对性地解决 BBR 在 STARTUP 阶段存在的性能问题，只是单纯地提高了整体的吞吐量。

腾讯公司开发的 TCPA 单边加速协议通过将初始窗口增大 2 倍来提高初始传输性能，但过度增大窗口并不适用于校园网络。校园网服务器的带宽低，单位时间内的处理能力弱，使用 TCPA 协议会破坏数据流分配的公平性。锐速加速算法通过过度补偿来提高传输性能，但在传输初期基本不会丢包，过度补偿的数据最后都会被丢弃，这大大浪费了本就不宽裕的校园网带宽。

本文针对 BBR 在 STARTUP 阶段存在的空窗问题，设计了更为平滑的时延估计器、STARTUP 阶段状态机和发送方流量状态机。通过改善估计器的加权系数，并引入瞬时平均偏差作为振荡平滑因子，以应对因未收到 ACK 而引起的时延振荡。为了补偿不可避免的空窗问题和序号回绕，在发送端设计流量状态机和 STARTUP 状态机来维持较高的链路吞吐量。实验结果表明，改进后的 BBR 在 STARTUP 阶段具有更好的传输性能。

2 BBR 在 STARTUP 阶段存在的问题

BBR 设置的接收窗口大小增益公式为：

$$cwnd(t)=2^t \tag{1}$$

其中，t 为往返时间(Round-Trip Time，RTT)周期轮次。

理想情况下，同一时刻链路上发送窗口和接收窗口的大小相等时，链路带宽的利用率最高。因此，设置每个 RTT 轮次内传输数据量的函数为：

$$data\_flight(t)=2^t \tag{2}$$

因为 t 为一个 RTT 周期，即传输时间为 1，单位周期内传输数据的速度等于传输的数据量。对传输速度进行积分，得到前一周期内传输的数据包数量：

$$data\_flight(t-1)=\int_{t-1}^{t} data\_flight(t)\mathrm{d}t=\frac{2^t}{2\ln 2} \tag{3}$$

BBR 中规定本轮的传输速度为前一轮的传输速度乘以增益系数，本轮管道中正在传输的数据包是上一个轮次节点发送数据与增益系数的乘积，得到如下关系：

$$bbr\_high\_gain * data\_flight(t-2)=data\_flight(t-1) \tag{4}$$

得到发送窗口的增益系数为：

$$bbr\_high\_gain=\frac{2}{\ln 2} \tag{5}$$

数据发送窗口的起始大小为 1，增益系数为 2.89；接收(拥塞)窗口的初始大小为 7，增益系数为 2[3]。在传输开始初期，会出现两者大小不匹配的情况，如图 1 所示。

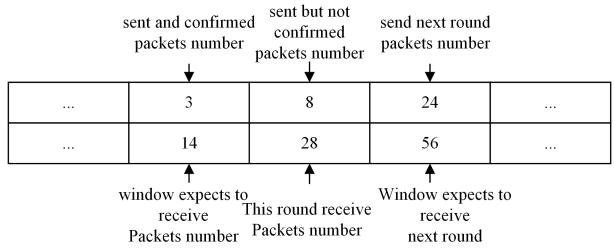


图 1 STARTUP 阶段窗口的示意图

Fig. 1 Schematic diagram of STARTUP phase window

图 1 中，上半部分为一个周期轮次内发送窗口发送的数据包数量，下半部分为接收窗口一个周期轮次内告知发送端可接收的数据包数量，同一轮次内的发送窗口小于接收窗口。接收端向发送端回复 ACK 报文时，会附带接收数据包的时间戳，发送端根据数据包发送和接收的时间戳差值计算 RTT，当规定时间内没有收到指定序号的 ACK 报文时，该序号分组测量的往返时延被记录为预设的最大值。

BBR 通过经典方法[15]计算得到平滑的 RTT 估计值，这种方法被称为指数加权移动平均法，其计算式如下：

$$srtt \leftarrow \alpha(srtt)+(1-\alpha)RTT_S \tag{6}$$

其中，$srtt$ 为 RTT 估计值，RTT_S 为新的 RTT 测量值，$\alpha=0.75$ 为加权因子[15]。在传输初期，往返时延一般很短，一旦出现规定时间内未确认的数据包，该次测量的 RTT 就会变为预设最大值，即便下一次测量到了已确认的数据包，RTT 估计值也会维持在一个很高的水平。BBR 通过探测时延来推测链路的可用带宽，如果预测的时延值较大，则会认为链路上已经没有更多的可用带宽或已经发生拥塞，从而进入排空阶段，等待未确认的数据包全部确认[3]，即空窗现象。

校园共享数据中心的简易图如图 2 所示，校园网数据传输时间一般较短，且带宽上限较低[16]。

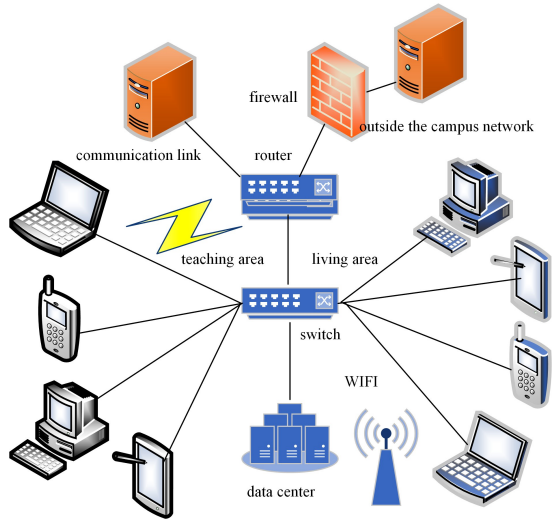


图 2 校园共享数据中心简易图

Fig. 2 Simple diagram of campus shared data center

空窗现象在没有发生拥塞的情况下会造成带宽闲置，且这种异常是传输协议本身的缺陷造成的。空窗现象几乎不可避免，如果大量正在运行的 BBR 流都遭遇了空窗，则会浪费大量的带宽，引发全局速度下降。大部分校园网用户使用无线网络与数据中心连接，无线网络的不稳定性也会在一定程度上加剧空窗的发生。如何提高校园网的带宽利用率是提高用户体验的重中之重。

3 单边适应算法设计

由前文的分析可以得到造成空窗问题的原因：发送窗口和接收窗口大小的增益系数为固定常量，无法根据实际情况进行自我调节，使得在初始阶段发送窗口与接收窗口的大小不匹配，连续的 RTT 测量值变化较大，时延估计器无法得到合适的期望值。针对这个问题，本文主要做了两方面的优化：1）设计更平滑的时延估计器，使得在 RTT 变化较大的情况下也可以得到合适的 RTT 估计值；2）若规定时间内未收到已发送数据包的 ACK 报文，发送窗口就无法右移，发送方需要根据流量数据的状态反馈做出相应的传输行为，避免带宽闲置。

3.1 更平滑的时延估计器

如果将 RTT 测量过程看作一个统计过程，测量均值或标准差可以更好地得到估计值。平均偏差可以很好地逼近标准差，且计算方法更便捷。因此，引入测量值与原值的平均偏差 $rttvar$ 作为振荡平滑因子，再引入两个新变量，即瞬时平均偏差估计值 $mdev$ 和预设的瞬时平均偏差最大值 $mdev\_max$，用于实现 $rttvar$ 的值会随着每个 ACK 报文的到来而更新。

使用偏差估计器计算瞬时平均偏差，计算式如下：

$$mdev=mdev*\alpha+(rtt_s-srtt)*\beta \tag{7}$$

其中，$mdev$ 预设为 0，rtt_s 为 RTT 测量值，$srtt$ 为 RTT 估计值，α 和 β 为加权因子。

$$mdev\_max=\max(mdev\_max,mdev) \tag{8}$$

其中，预设 $mdev\_max$ 为 50 ms。

$$rttvar=mdev\_max \tag{9}$$

引入瞬时平均偏差后，原有时延估计器变为更平滑的时延估计器：

$$srtt=\alpha*srtt+\beta*rtt_s-rttvar \tag{10}$$

其中，$rttvar$ 为振荡平滑因子。

图 3 为更平滑的时延估计器的设计架构图，每个新测试得到的 RTT 值都会被送入时延估计器。

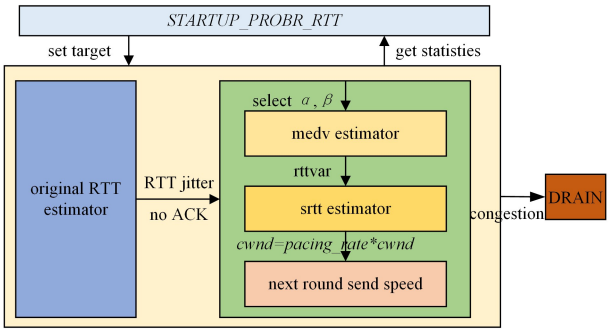


图 3 更平滑时延估计器的设计架构图

Fig. 3 Design architecture diagram of a smoother delay estimator

原始的时延估计器会对抖动的输入值做出错误判断，这时就需要使用全新的时延估计器。输入值首先进入瞬时平均偏差估计器，将估计值通过比较函数转化为振荡平滑因子，再输入时延估计器得到合理的时延反馈。每一轮的时延反馈都将作为下一轮发送速度的凭据传输到发送方。如果时延估计器经过计算认为链路已经发生了拥塞，则会通知发送方进入排空状态。

3.2 发送方流量状态机和 STARTUP 阶段状态机

平滑时延估计器可以减少等待重传现象的发生，但发送窗口与接收窗口大小不匹配的情况不会消失，即无法完全保证在 STARTUP 阶段发送的数据都能在规定时间内收到 ACK 报文。在未收到 ACK 报文时发送窗口无法右移，即带宽依然会出现闲置。这就需要对发送方的行为进行分类，以便于算法可以继续发送数据而不是等待已发送数据包的 ACK 到来。

由图 4 中的例子可以看到，在第二个 RTT 周期，发送窗口的大小为 3，接收窗口的大小为 14，序号为 1 的数据包已经被发送并确认，序号为 2，3，4 的数据包正在管道中传输未被确认，接收窗口还可以接收序号 5－15 的数据包，序号大于 15 的数据包暂时不能被接收。如果接收端按顺序接收数据包，会递增 rcv_next。接收方会通知发送方窗口向右移动，递增 rcv_adv。发送方流量状态机需要在未收到 ACK 的情况下递增 $send\_max$ 和 rcv_next，即将窗口指针右移。这期间需要保证，增发数据的 ACK 为有效数据包确认，即增发数据不超过剩余可用窗口。

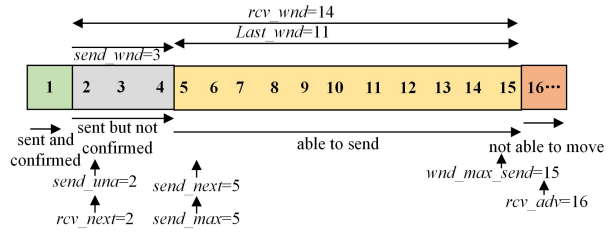
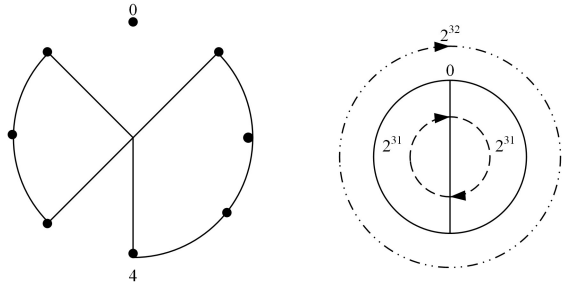


图 4 发送和接收序号空间举例

Fig. 4 Example of sending and receiving sequence number space

如图 5 所示，序号长度为 3bit、发送窗口大小为 4 的例子中以 0 作为基准序号，通过二进制的补码计算和回绕机制[17]，可以得到这样的结论：序号 5，6，7 位于序号 0 的左侧，对应数据包的到达时间早于序号 0，即对应的数据包比序号为 0 的数据包先到达；序号 1，2，3，4 位于序号 0 的右侧，对应数据包的到达时间晚于序号 0，即对应的数据包后到达。



(a) 3 bit 样例 (b) 实际序号分布

图 5 TCP 数据包序号比较

Fig. 5 Comparison of TCP data packet sequence numbers

图 5(a)代表的实际含义为此时发送方收到序号为 0 的数据包的确认报文，如果再收到序号为 5,6,7 的确认报文就会认定其为已经被接收而直接将其丢弃，但会等待序号为 1,2,3,4 的数据包确认报文到达。实际的 TCP 传输中序号长度为 32bit，如图 5(b)所示，可以有 2^{32} 个数据序号且起始序号随机。因此，BBR 算法需要识别不同 ACK 带回的状态反馈。

如图 4 所示，若数据包是按顺序依次传输完成的，*send_next* 和 *send_max* 会指向同一个数据包，如果重传定时器超时，发送方会重传数据，*send_max* 是已发送过的最高序号，*send_max* 会被设为 *send_nua*，即 *send_next* 小于 *send_max*，指向窗口左侧。增发数据的确认报文可能会比先发送数据的确认报文先回到发送方，这与重传数据返回的确认情况相同。因此，在设计状态机时，要加上序号比较，以免发送方认为未确认的先前数据已经被接收而丢弃数据。

如图 6 所示，将 STARTUP 阶段分为 3 个状态：GOOD，NORMAL，BAD。在 GOOD 阶段，发送的数据包都能在规定时间内收到 ACK，且往返时延保持在很低的水平，发送方认为链路具有更多的可用带宽，会不断加快发送速度。数据具体的传输过程无法准确得知，但会出现偶然的数据包或 ACK 报文丢失或乱序，测试时延发生抖动，发送方会收到 SACK 报文。NORMAL 阶段具有一定的容错性，对整个传输过程不会产生太大的影响。BAD 阶段分为两种情况，第一种是空窗现象，如果规定时间内无法收到 ACK 报文，时延估计器无法得到合适的估计值，发送方得到错误反馈进入排空状态。这种情况需要继续增发新数据，窗口右移，然后等待已发送数据的 ACK 到来。第二种情况是前方发生了拥塞，后发送的数据都被丢弃，这就需要算法从 STARTUP 阶段进入 DRAIN 状态。发送方降低发送速度并重传丢失数据，直至链路恢复正常。

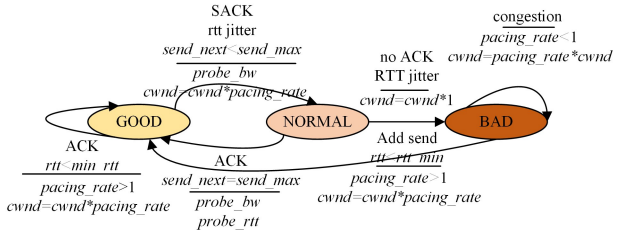


图 6 STARTUP 阶段的有限状态机

Fig. 6 Finite state machine in STARTUP phase

图 7 给出了发送方流量有限状态机，将外流量分为 6 个状态：new，blocked，waiting，time_waiting，running，terminated。在 new 状态的流量还未发送时，发送窗口右移后，流量会进入 running 状态，转化为在管道中飞行但还未确认的外流量。当发送方收到外流量的 ACK，该流量就会进入 terminated 状态，表示已经完成传输。若发送方没有接收到外流量的 ACK，则会进入 waiting 状态，等到确认报文的到来。这个状态又分为 3 种情况：如果在规定时间内收到 ACK，则进入 terminated 状态完成传输。如果出现空窗现象，则进入 time_waiting 状态，窗口右移，发送方发送全新的数据。如果此时算法认为发生拥塞，流量就会进入 blocked 状态，停止新的发送，等待已发送的数据全部确认。如果拥塞越来越严重，直至发生缓冲区溢出，则会进入 terminated 状态，强制终止传输。

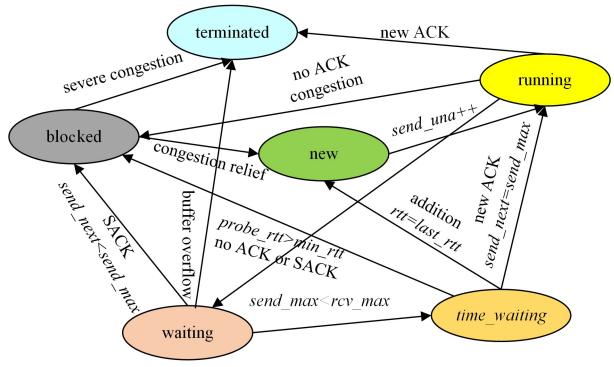


图 7 发送方流量有限状态机

Fig. 7 Sender traffic finite state machine

4 实验过程与结果

本实验在南京邮电大学校内无线局域网 NJUPT 下进行，直接在 Linux 内核中修改 BBR 协议进行测试，设备的具体参数如表 1 所列。

表 1 设备的具体参数

Table 1 Equipment specific parameters

operating system	ubuntu16.04
kernel	4.15.1
RAM/GB	4.1
disk/GB	210.2
processor	Inter(R) Core(TM) i7-5500U CPU@2.40GHz

4.1 实验参数选取

4.1.1 时延估计器加权系数选择

Linux 的时钟粒度是 1 ms，刷新速度极快。BBR 算法依靠时间戳计算 RTT 值，会对每一个完整传送的包进行计算，快速传输时窗口会通过很多包。如果只是采用平均偏差，大量的样本值会让平均偏差变小，失去了平滑时延的作用。又因为每一个样本值都存在差异，所以需要引入瞬时平均偏差随时钟一起快速刷新，保持对 RTT 变化的敏感度。但如果使用该算法的操作系统的时钟粒度较大，平均偏差就可以达到平滑的作用。

式(7)中，α 为原有平均偏差估计值的加权系数，β 为新测量的平均偏差的加权系数，二者被存储在路由表的记录项中，随着操作系统的时钟不断被刷新。加权系数越大，对应的平均偏差值对估计值的影响就越大。STARTUP 阶段的空窗现象会使 RTT 测量值突然增大，新测量的平均偏差瞬间变大，进而使估计值过度增大。因此需要减小 β 的值，以降低平均偏差抖动对估计器输出值的影响。

α，β 实际反映了估计器对旧值与新值的敏感程度，但 β 值过低会降低估计器对时延变化的敏感性。TCP 设计规范[RFC793]中推荐的 α，β 值分别为 0.9 和 0.1。但 Jacobson 已经详细论证了这个配比在 RTT 范围变化较大的情况下无法跟上 RTT 的变化[15]。原有的 BBR 算法采用了 Jacobson 的配比系数，$\alpha=0.75$，$\beta=0.25$，却无法避免空窗问题。因此，本文选取折中的加权系数，$\alpha=0.8$，$\beta=0.2$。

对同一次传输进行检测，采集 19 个样本点，得到图 8 所示的不同加权系数的时延估计值。原有的 BBR 算法采用

雅各布森设计的经典加权系数：$\alpha=0.75$，$\beta=0.25$，改进后的算法添加了瞬时平均偏差估计器，两个估计器都将加权系数改为：$\alpha=0.8$，$\beta=0.2$。由图 8 可以看出，两个时延估计器都可以对实际时延进行平滑操作，但改进后的算法得到的时延曲线更为平滑。从表 2 中可以看出，时延的平均偏差从 0.74 降到 0.33 再降到 0.21，即改进后的算法时延分布更平滑。从平均数和中位数数据来看，改进的算法对时延的估计精度分别提高了 2.5%和 1%。

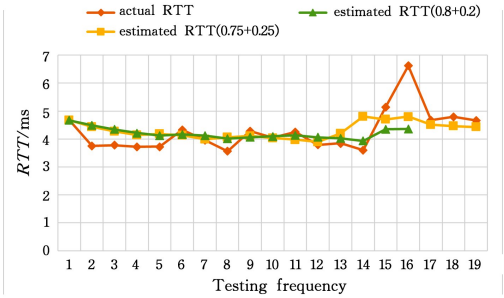


图 8 不同加权系数时延估计值

Fig. 8 Delay estimates with different weighting coefficients

表 2 不同时延估计器的数据

Table 2 Different time delay estimator data

	actual RTT	$\alpha=0.75,\beta=0.25$	$\alpha=0.8,\beta=0.2$
stdev	0.736706	0.326381006	0.206932
avg	4.256316	4.343684211	4.238947
median	4.05	4.2	4.16

4.1.2 振荡平滑因子的效果

当 RTT 测量值维持在正常的时延值时，*mdev* 值很小，相应的 *mdev_max* 就会维持在预设值。预设值 50 ms 远低于 BBR 时延探测的最低时限 200 ms，因此正常情况下振荡平滑因子比时延值小得多，不会对 *srtt* 造成很大的影响，可以满足预设要求。

当 RTT 测量值突然变大时，测量值和现有值的差值也会变大，*mdev* 变大并超过 *mdev_max*，*rttvar* 的值也随之增大，从式(9)和式(10)中可以看到，*rttvar* 变大降低了 *srtt* 的估计值，缓冲了时延测量值变化过大的影响，可以满足预设要求。

4.1.3 状态机复杂性分析

本文所述算法只应用在发送端，因此不受网络中协议和终端应用程序的影响，搭建方便。BBR 算法本身的算法复杂性和体量远低于现在主流的 CUBIC 算法。改进后的算法在状态机上的优化属于对不可避免的空窗问题的带宽补偿，并不会增加算法的复杂性。这是因为传输层协议并不理会数据在链路中的具体传输，其只负责识别和控制传输状态。详尽准确的状态控制对下层网络架构是良性的，可以提高底层协议的效率。

4.2 传输性能对照

4.2.1 带宽与公平性对照

本文中的图表统一使用 bbr 来代表原有的 BBR 算法，使用 bbr2 来代表改进的 BBR 算法。从图 9 可以看出，在传输的前 50 s 内，改进后的 BBR 算法的带宽曲线大部分在原有 BBR 算法曲线的上方，改进后的算法可以在相同的条件下获得更多的带宽。

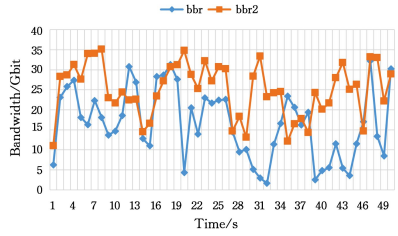


图 9 带宽对比

Fig. 9 Bandwidth comparison

随着网络的发展，不同流接收到的带宽会出现越来越不平等的情况[17]，因此公平性是现实世界中评价拥塞控制协议最重要的标准之一[18]。

在同一次传输任务中同时启用 15 个并行流，得到图 10 和表 3 中的结果。对单次传输任务来说，并行流越多，带宽损耗越严重[19]。从传输流带宽的总和、中位数和平均值来看，改进后的算法可以获得更多的带宽。从标准差数据和图 10 中的带宽分布也可以看出，改进后的算法各个流所占的带宽资源更平均，数据流之间的公平性更好。

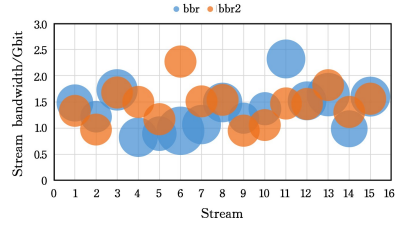


图 10 并行流占用带宽分布

Fig. 10 Bandwidth distribution occupied by parallel streams

表 3 公平性指标

Table 3 Fairness indicators

bandwidth/Gbit	bbr	bbr2
sum	20.3	21.6
stdev	0.382279	0.329312
median	1.37	1.47
avg	1.352533	1.4404

4.2.2 单次传输性能对照

在 HTTP 协议下，分别使用改进前后的 BBR 算法传输 47 MB 的文件，以比较性能变化。

首先对原有 BBR 算法进行测试，从 Wireshark 的专家信息中可以发现有一些位于 STARTUP 阶段未捕获的 ACK 片段，如图 11 所示。

2	[TCP ACKed unseen segment] 80 → 39084 [ACK] Seq=1 Ack=2 Win=64240 Len=0	Sequence	TCP
5	[TCP ACKed unseen segment] 80 → 35776 [ACK] Seq=1 Ack=2 Win=64240 Len=0	Sequence	TCP
6	[TCP ACKed unseen segment] 80 → 59044 [ACK] Seq=1 Ack=2 Win=64240 Len=0	Sequence	TCP
8	[TCP ACKed unseen segment] 80 → 47492 [ACK] Seq=1 Ack=2 Win=64240 Len=0	Sequence	TCP
4225	[TCP ACKed unseen segment] 80 → 39640 [ACK] Seq=1 Ack=2 Win=64240 Len=0	Sequence	TCP
4226	[TCP ACKed unseen segment] 80 → 50794 [ACK] Seq=1 Ack=2 Win=64240 Len=0	Sequence	TCP
4635	[TCP ACKed unseen segment] 80 → 41624 [ACK] Seq=1 Ack=2 Win=64240 Len=0	Sequence	TCP

图 11 STARTUP 阶段未捕获的 ACK 片段

Fig. 11 Uncaptured ACK fragments in STARTUP phase

从专家信息中可以发现，序号为 2，5，6，8 的数据包没有收到 ACK 报文段，其下一个序号的恢复时间指标(Recovery Time Object，RTO)突然增大很多。序号为 2 的数据包对应的 RTO 值为 0.0002，序号为 3 的变为了 2.0480；序号 5 和序号 6 对应的值分别为 2.0482 和 2.0483，序号 7 对应的值变为 4.0966。这说明用来计算 RTO 的时延估计器得到了错误的期望值，未收到 ACK，RTT 估计值会陡增，导致 RTO 突然变大。图 12 中，Time 列表显示了 RTO 值，由时延估计值计算求得。

No.	Time	Source	Destination	Protoc	Length	Info
1	0.0000…	192.168.174.1…	203.208.41.97	TCP	54	39084 → 80 [ACK] Seq=1 Ack=1 Win=32256 Len=0
2	0.0002…	203.208.41.97	192.168.174.1…	TCP	60	[TCP ACKed unseen segment] 80 → 39084 [ACK] Seq=1 Ack=2 Win=64240 Len=0
3	2.0480…	192.168.174.1…	216.58.200.240	TCP	54	35776 → 80 [ACK] Seq=1 Ack=1 Win=43800 Len=0
4	2.0481…	192.168.174.1…	180.163.150.3	TCP	54	59044 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
5	2.0482…	216.58.200.240	192.168.174.1…	TCP	60	[TCP ACKed unseen segment] 80 → 35776 [ACK] Seq=1 Ack=2 Win=64240 Len=0
6	2.0483…	180.163.150.3	192.168.174.1…	TCP	60	[TCP ACKed unseen segment] 80 → 59044 [ACK] Seq=1 Ack=2 Win=64240 Len=0
7	4.0966…	192.168.174.1…	101.91.79.35	TCP	54	47492 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
8	4.0968…	101.91.79.35	192.168.174.1…	TCP	60	[TCP ACKed unseen segment] 80 → 47492 [ACK] Seq=1 Ack=2 Win=64240 Len=0

图 12 改进前 Wireshark 抓包数据的部分列表

Fig. 12 Partial list of Wireshark captured data before modification

表 4 传输会话信息

Table 4 Transmission session information

	packets	Bytes/M	packetsA-B	bytes A-B/K	packets B-A	bytes B-A/M	duration
bbr	4391	47	1954	106	2437	47	12.069
bbr2	4615	47	2047	111	2568	47	8.2746

从表 4 中可以看到，改进后的算法传输的数据包数量从 4391 增长到了 4615；传输持续时间则从 12.069 s 缩短为 8.2746 s。说明改进后 BBR 的传输速度加快，单位时间内占有的带宽增多。

统计 BBR 算法改进前后，传输的数据包长度如表 5 所列。

表 5 改进前后数据包的长度统计

Table 5 Data packet length statistics before and after modification

Packet Lengths	bbr			bbr2		
	bbr	avg	Percent/%	bbr2	avg	Percent/%
40～79	2111	54.61	45.54	2155	54.4	45.09
80～159	23	113.48	0.50	25	109.28	0.52
160～319	15	257.07	0.32	17	251.82	0.36
320～639	34	502.91	0.73	10	517.1	0.21
640～1279	37	890.89	0.80	36	943.56	0.75
1280～2559	368	1525.95	7.94	409	1520.73	8.56
2560～5119	181	3439.45	3.91	192	3390.73	4.02
5120 and greater	1866	25027.8	40.26	1935	24079.55	40.49

典型的 TCP 控制数据包大约是 54 byte，两次实验长度在 40～79 字节的数据如此之多，说明承载本次传输的是 TCP 协议。从表 5 可以发现，大长度数据包的数量均有增加，由图 13 可以更清晰地看出：长度超过 1280 字节的部分，改进后所占的百分比都超过了 50%。

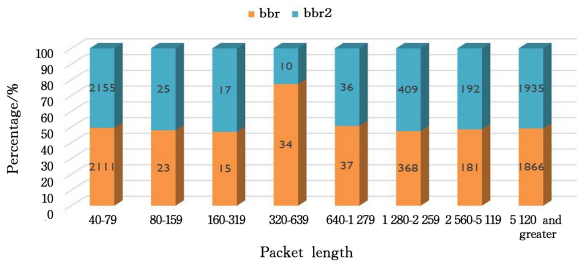


图 13 改进前后数据包的长度分布

Fig. 13 Data packet length distribution before and after modification

从图 13 可知，改进后长度较长的数据包所占的比例增多，说明数据的传输速度和效率相比改进前有所提高。图 14 中，未改进前，在传输起始处几乎没有数据包传输，在 5 s 后，传输数据才开始增长，在 19 s 结束传输任务。改进后，在第 2 s 就有数据传输，从第 4 s 开始，数据传输速度快速增长，从第 6 s 开始维持在很高的水平，第 15 s 传输任务结束。可以得出结论：改进后，算法的启动速度和整体传输速度加快，提高了带宽利用率。

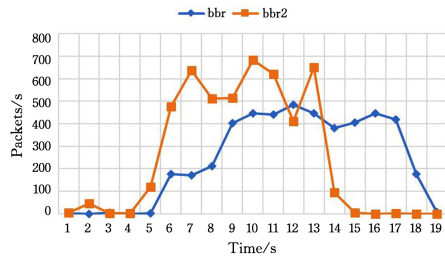


图 14 改进前后的 I/O 流量图

Fig. 14 I/O flow diagram before and after modification

图 15(a)和图 15(b)为 tcptrace 图，如果出现了平台期，则说明这段期间没有新的数据包发送。图 15(a)中出现了大量平台期，0～2.5 s 内至少有一半的时间没有发送新的数据，而是在等待已发送数据包的 ACK 的到来。图 15(b)同样存在平台期，但平台期持续时间非常短，这说明在 STARTUP 阶段出现平台期是时延估计器得到了错误的估计值，而不是丢包引起的，继续增发新数据也没有给瓶颈带宽造成额外的压力。图 15(c)和图 15(d)为吞吐量图，曲线表示吞吐量随时间的变化过程，蓝点表示段长度，蓝点越密集吞吐量就越大，

曲线为吞吐量变化曲线。图 15(c)中吞吐量曲线成“丘陵”状，在 1.4～2 s 处甚至出现了吞吐量下降的现象。图 15(d)中吞吐量曲线在 0～1.1 s 间稳步增长，1.1s 后维持在稳定数值上下。图 15(d)中的点明显比图 15(c)中的点密集，这说明改进后算法的吞吐量能够快速增长并达到稳定，管道内传输的数据包增多了。

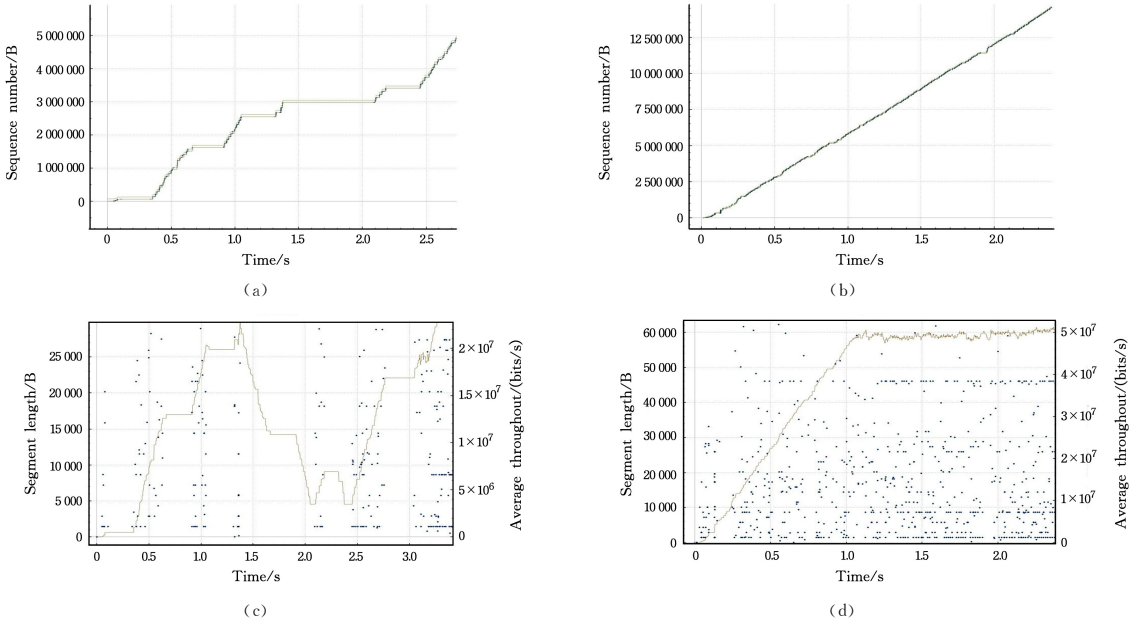


图 15 改进前后的 tcptrace 图和吞吐量图比较(电子版为彩色)

Fig. 15 Comparison of tcptrace graph and throughput graph before and after modification

4.2.3 与其他主流算法的性能进行比较

由于被动拥塞控制算法具有自适应 AIMD 参数[20]，相比 BBR 可以有效地减少空窗现象的发生。因此，本文增加了与被动拥塞控制算法(Reno 和 CUBIC)的性能比较。

图 16 给出了不同算法的带宽比较结果，可以看出，在存在空窗现象的前 10 s 传输中，Reno 和 CUBIC 算法出现了典型的锯齿曲线，原有的 BBR 算法从第 4 s 开始带宽占有量持续下跌，丧失了 BBR 算法最大的优势，即尽可能多地占用带宽。改进后的 BBR 算法仅在第 8 s 经历了一次带宽抖动，但很快恢复了极高的带宽占用量。改进后的 BBR 算法的整体曲线最为平滑，说明该算法的鲁棒性最好，可以在非拥塞的情况下应对其他因素的干扰。

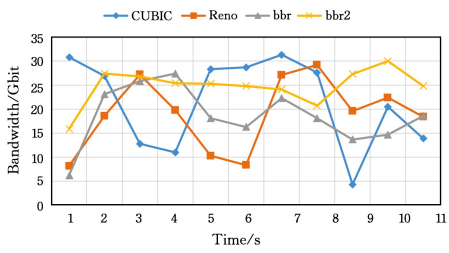


图 16 不同算法的带宽比较

Fig. 16 Comparison of bandwidth of different algorithms

表 6 列出了不同算法的公平性数据，在 5 个并行流的情况下，改进后的 BBR 占用总带宽最多，Reno 最少，CUBIC 和原有 BBR 占用带宽相近。从各个流带宽标准差来看，原有的 BBR 算法的各个流的标准差为 1.1934，数值最大，占用带宽最不平均，公平性最差。CUBIC 算法各个数据流的标准差为 0.5343，数值最小，占用带宽最平均，公平性最好。改进后的 BBR 算法的标准差为 0.9978，略好于 Reno 算法和改进前的 BBR 算法。

表 6 不同算法的公平性数据

Table 6 Fairness data of different algorithms

Gbit	Reno	CUGIC	bbr	bbr2
stream 1	4.88	5.01	4.04	4.75
stream 2	4.24	4.03	4.36	7.3
stream 3	3.19	4.59	4.94	4.73
stream 4	3.45	5.35	4.17	5.69
stream 5	6.07	5.5	7.26	4.75
sum	21.8	24.5	24.8	27.2
stdev	1.0398	0.5343	1.1934	0.9978

在校园网络中，获得更多的带宽极为重要。综合资源占用率(带宽占有量)和公平性来说，本文算法尽管公平性弱于 CUBIC 算法，但大幅提升的带宽可以显著提升用户体验。后续研究可以考虑如何提升 BBR 算法的公平性，进一步优化 BBR 算法的传输性能；另外，可通过设计一套平滑的自适应增益机制(AIMD 的锯齿图存在过度衰减的问题)，来增强 BBR 算法的普适性。

结束语 校园网中使用的 BBR 算法在 STARTUP 阶段存在空窗问题。本文通过改善估计器的加权系数，并引入瞬时平均偏差作为振荡平滑因子，以应对因未收到 ACK 而引起的时延振荡。为了尽可能解决不可避免的空窗问题和序号回绕，在发送端设计流量状态机和 STARTUP 状态机来维持较高的链路吞吐量。

实验证明，改进后的 BBR 算法的各项传输性能均优于原始的 BBR 算法。从整体性能考虑，改进后的 BBR 算法优于主流的被动拥塞算法，是提升校园网传输性能的最优选择。

目前，BBR 仍不是主流的拥塞控制算法，该算法缺失类似于 AIMD 的自适应机制。脱离校园网的应用场景，本文

算法的性能优越性会有所下降。

参 考 文 献

[1] FAN Z F, LI S, ZHANG D. SDN data center network congestion control algorithm based on traffic scheduling[J]. Computer Science, 2017, 44(S1): 266-269, 273.

[2] WANG H H, ZHOU Y W, LIU J B. Hybrid-based routing algorithm for network congestion control in LLN[J]. Computer Science, 2019, 46(6): 107-111.

[3] NEAL C, CHENG Y C, STEPHEN G, et al. BBR: congestion-based congestion control[J]. Communications of the ACM, 2017, 60(2): 58-66.

[4] ZHANG H, ZHU H, XIA Y, et al. Performance Analysis of BBR Congestion Control Protocol Based on NS3[C]// 2019 Seventh International Conference on Advanced Cloud and Big Data (CBD). Suzhou: IEEE CS CPS, 2019: 363-368.

[5] JAMES G, KATHLEEN N. Bufferbloat: dark buffers in the Internet[J]. Communications of the ACM, 2012, 55(1): 57-65.

[6] HORIE Y, THI D, NGUYEN A, et al. A Comparison of Congestion Control Algorithms in Emulated Wi-Fi Networks[C]// 2020 International Conference on Information and Communication Technology Convergence (ICTC). Guayaquil: Association for Computing Machinery, 2020: 305-310.

[7] KANAYA T, TABATA N, YAMAGUCHI S. A Study on Performance of CUBIC TCP and TCP BBR in 5G Environment [C]// 2020 IEEE 3rd 5G World Forum (5GWF). Bangalore: IEEE Press, 2020: 508-513.

[8] SASAKI K, YAMAGUCHI S. A Study on Bottleneck Bandwidth Estimation Based onAcknowledge Reception on TCP BBR [C]// 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC). Madrid: IEEE Computer Society, 2020: 1107-1108.

[9] NEIL A, MATTEO V, ANDRIUS A, et al. Mind the delay: the adverse effects of delay-based TCP on HTTP[C]// Proceedings of the 16th International Conference on emerging Networking Experiments and Technologies (CoNEXT '20). New York: Asso-ciation for Computing Machinery, 2020: 364-370.

[10] ZAD D, AHMED F, SHARMA P, et al. Homa: An Efficient Topology and Route Management Approach in SD-WAN Overlays [C]// IEEE INFOCOM 2020 — IEEE Conference on Computer Communications. Beijing: IEEE Press, 2020: 2351-2360.

[11] LI Y L, MIAO R, LIU H Q, et al. HPCC: high precision congestion control[C]// Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19). New York: Association for Computing Machinery, 2019: 44-58.

[12] MA Y, LIANG J. Throughput Maximization of NFV-Enabled Multicasting in Mobile Edge Cloud Networks[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(2): 393-407.

[13] CHEN K F, SHAN D F, LUO X H, et al. One Rein to Rule Them All: A Framework for Datacenter-to-User Congestion Control[C]// 4th Asia-Pacific Workshop on Networking (APNet '20). New York: Association for Computing Machinery, 2020: 44-51.

[14] FOREMCICH A, SNOEREN A, PORTER G, et al. Corundum: An Open-Source 100-Gbps Nic[C]// 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). IEEE Press, 2020: 38-46.

[15] JACOBSON V. Congestion avoidance and control[J]. ACM SIGCOMM Computer Communication Review, 1995, 25(1): 157-187.

[16] ZENG G X, HU S H, ZHANG J X, et al. Overview of Data Center Network Transmission Protocol[J]. Computer Research and Development, 2020, 57(1): 74-84.

[17] ASHUTOSH S, FRAIDA F, SHIVENDRA P. A Low Latency Congestion ControlThat Can Compete[C]// Proceedings of the Student Workshop(CoNEXT '20). New York: Association for Computing Machinery, 2020: 15-16.

[18] LLOYD B, GANESH A, ETHAN K, et al. On the Future of Congestion Control for the Public Internet[C]// Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20). New York: Association for Computing Machinery, 2020: 30-37.

[19] AYUSH M, SUN X P, ATISHYA J, et al. The Great Internet TCP Congestion Control Census[J]. Measurement and Analysis of Computing Systems, 2019, 3(3): 1-24.

[20] NANDITA D, TIZIANA R, CHENG Y C. An argument for increasing TCP's initial congestion window[J]. ACM SIGCOMM Computer Communication Review, 2010, 40(3): 26-33.

MA Li-wen, born in 1996, postgraduate, is a member of China Computer Federation. Her main research interests include computer network and network programming.

ZHOU Ying, born in 1978, Ph.D, associate professor. Her main research interests include networked control system and so on.

(责任编辑:喻藜)