初探拥塞控制与 BBR 算法

摘要

拥塞控制是计算机网络中的重要组成部分，并随着互联网的发展而不断显现其重要性。近代互联网的复杂度、联系紧密度都远超之前人们的想象，想管理好如此这样一个庞大系统，拥塞控制是必不可少的技术。拥塞控制确保了当网络出现拥塞时，网络仍然能够以较低的利用率运转，且各用户尽可能公平地利用剩余带宽，避免网络瘫痪。

本文通过拥塞控制的历史与简单的计算机通信模型，引入了拥塞控制原理与算法分析，并着重介绍了其中 BBR 算法的具体实现，分析其优劣与发展潜力。

关键词： 拥塞控制；BBR 算法

Abstract

Congestion control is an important part of computer networks and continues to show its importance with the development of the Internet. The complexity and tightness of the modern Internet is far beyond what was previously imagined, and congestion control is an essential technique for managing such a large system. Congestion control ensures that when the network is congested, the network can still operate at a low utilization rate, and each user can utilize the remaining bandwidth as fairly as possible to avoid network paralysis.

This paper introduces the analysis of congestion control principles and algorithms through the history of congestion control and a simple computer communication model, and focuses on the implementation of the BBR algorithm and analyzes its advantages and disadvantages as well as its development potential.

Key Words: congestion control; BBR algorithm

1 绪论

1.1 研究背景

随着互联网的普及和应用范围的不断扩大，网络中的信息呈现爆炸式增长，对网络及其基础设施提出了更高的要求。网络数据传输的条件也在不断变化：在地面互联网中，传输带宽越来越高，链路构成越来越复杂；而在一些特殊网络中，如卫星网络中，通信距离越来越远，传播延时越来越大。这都为当前的传输协议带来了前所未有的挑战[1]。

TCP 作为一种可靠传输协议，通过在数据传输中引入确认和重传机制保证可靠性。当链路不稳定时，丢包率和错误率都会上升；当网络拥挤时，排队的分组总量超出了路由器缓存的大小，导致后续到达的分组被丢弃。由此，大量数据需要重传，从而加剧了网络拥塞。

1983 年，TCP 被默认部署在当时的互联网—— ARPANet 上。1986 年 10 月，由于 ARPANET 网络中的一个路由器出现了故障，大量数据包被重复发送。在加州大学伯克利分校和 400 码外的劳伦斯伯克利国家实验室之间的 32 kbps 链路上检测到互联网拥塞崩溃，在此期间吞吐量下降了近 1000 倍，降至 40 bps[2]。这一现象引起了网络研究者的注意，因此提出了拥塞控制，使其成为一门新的研究领域。

拥塞控制算法是 TCP 协议中非常重要的一部分。在网络拥塞时，主机不再贪婪地发出数据，而是采取更加保守的策略，降低自身发送数据包的速率，从而保证互联网的可用性，尽可能提高带宽利用率。

1.2 主机通信模型

本文不考虑路由器的转发寻址功能，因此将复杂的计算机网络抽象为点对点的通信链路，如图 1 所示。

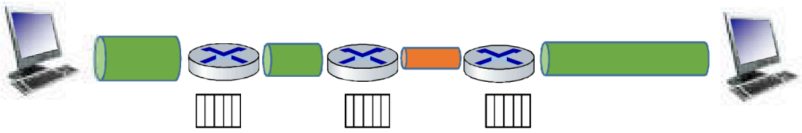

图 **1**　网络抽象模型

数据被划分成一个个分组在网络上传播。每个路由器具有一个缓存池，用于缓存路由器接收到的分组。若将链路当成水管，每个数据包分组作为水流，则缓存相当于水池，具有积蓄作用。当某个路由器的输入速率大于输出速率时，多余的分组将在缓存中排队。若分组到达路由器，而缓存已满，该分组将会被丢弃。

在点对点数据传输中所能达到的最高速率取决于瓶颈链路的带宽，也就是传输能力最弱的链路的带宽。

空闲网络中的分组增加，可以提高网络吞吐量。但是当拥塞发生时，网络中的每台主机都需要降低自身的数据发送速率。这必然会导致网络的总体利用率变低，就像在堵车时，道路的车流量实际是减少的。

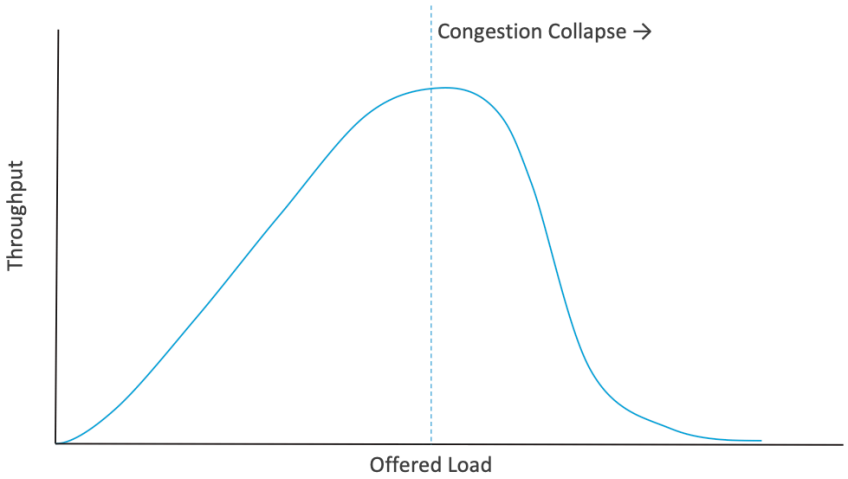

图 **2**　随着负载的增加，吞吐量会上升，然后在拥塞崩溃点下降。[3]

2 拥塞控制

2.1 拥塞控制概述

计算机与路由器构成了一个庞大的网络，任一时间任一节点的数据传输量都在变化。拥塞控制算法运行于主机之上，（在不改变现有网络协议与结构条件下）不能直接访问路由的缓存状态。因此拥塞控制算法需要通过其他办法探知网络的拥挤程度，并控制主机行为缓解拥塞。

拥塞控制的主要目的有两个：一是在网络不拥塞时，让数据发送得尽可能快；二是当拥塞发生时，主动降低数据发送速率，减少重传，避免发生网络崩溃。

除了探测拥塞、控制速度与重传，在拥塞控制算法的具体实现中，还需要考虑算法的公平性。拥塞发生时，网络资源不应被少量主机占有，而是需要尽可能地让所有主机共享相同的带宽。

2.2 拥塞模型

从 TCP 的角度来看，任意复杂的路径都表现为具有相同 RTT（Round-trip time，往返时间）和瓶颈速率的单个链路。RTprop（Round-trip propagation time，往返传播时间）和 BtlBw（Bottleneck Bandwidth，瓶颈带宽）这两个物理约束限制了传输性能。如果网络路径是物理管道，则 RTprop 将是其长度，BtlBw 将是其最小直径[4]。

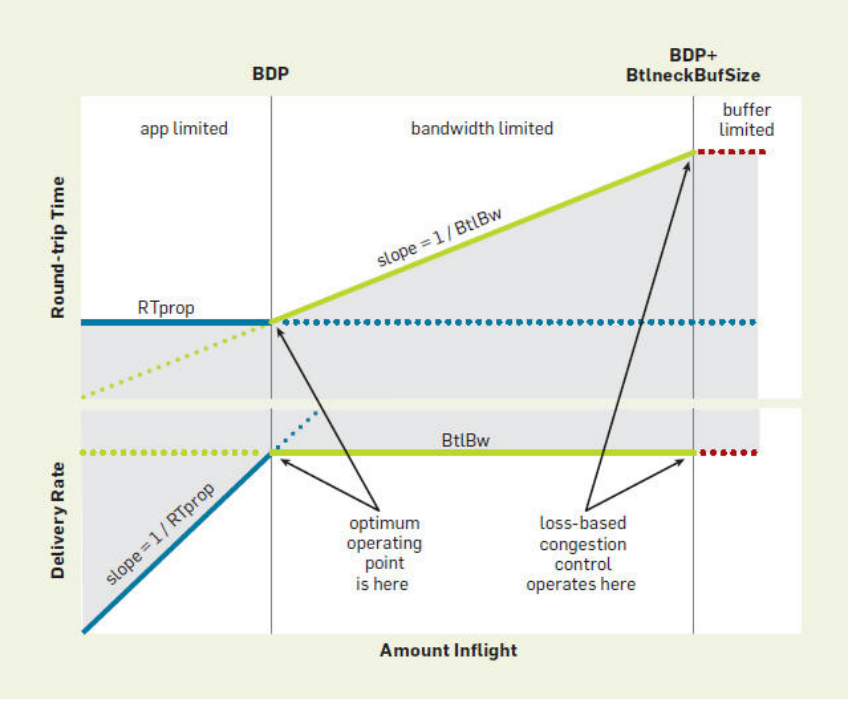

图 3　传输速率与往返时间的关系[4]

图 3 展示了网络运行过程中的三个阶段。第一个阶段，网络处于轻载状态，由于分组无需排队，往返延迟不变，而吞吐量完全取决于主机的发送速率，发送方发送得多快，接收方就能收得多快。第二个阶段，应用进程向网络中的注入速率超过了瓶颈链路的带宽，这个带宽本质上是吞吐量的上限。分组在缓存中排队，而往返延迟随排队的长度增

加而增加。此时网络已经出现了拥塞。第三个阶段，网络负载继续增加，排队分组数量超过了缓存容量，多余分组被丢弃，数据需要重传，因此往返延迟与吞吐量是未知的。

2.3 拥塞控制算法

根据拥塞控制算法的网络感知方式，大致将其分为两类。

2.3.1 基于丢包的拥塞控制算法

基于丢包的拥塞控制算法（Black box algorithms[5]）又称经典拥塞控制算法。此类算法主要通过是否发生丢包来探测拥塞，采用负反馈方式对主机进行控制。常见的有 Tahoe 算法、Reno 算法、NewReno 算法、SACK 算法，CUBIC 算法等。目前家用 Windows 与 Linux 系统采用 CUBIC 算法作为其默认拥塞控制算法。

此类算法的最大问题是，需要网络实际上出现丢包后，算法才能作出响应。此时网络运行处于第三个阶段，网络的可用性已经大幅下降。同时该类算法还有一些其他问题，例如无法分辨由于高出错率造成的丢包，导致不应有的降速。在无线传输领域，链路的出错率高，经典拥塞控制算法的表现与基于延迟的拥塞控制算法产生了较大差距。

经典拥塞控制算法的工作优势区间是链路多，带宽小，路由器缓存小的网络。然而近年来，随着网络基础设施的不断升级，缓存增大，此类算法在实际应用上的优势已经不明显。

2.3.2 基于延迟的拥塞控制算法

基于延迟的拥塞控制算法（Grey box algorithms）通过主动探测 RTT 来感知拥塞。常见的有 Vegas 算法、BBR 算法等。

基于延迟的拥塞控制算法能够在网络运行的第二个阶段感知到拥塞并及时反馈，解决了经典拥塞控制算法的最大痛点。由于其探知拥塞的时机较早，可以很好地适应网络的动态变化。

基于延迟的拥塞控制算法在丢包率高时具有极佳的表现。其中，Vegas 算法过于温和，在面对经典拥塞控制算法时难以取得足够的带宽。而 BBR 算法则具有不逊色于 CUBIC 算法的竞争力，因此成为最具有潜力的拥塞控制算法。

3 BBR 算法

3.1 算法概述

BBR（Bottleneck Bandwidth and Round-trip propagation time）拥塞控制算法由 Google 在 2016 年提出，其通过实时测量网络瓶颈带宽和最小延迟，计算带宽延时积（Bandwidth-Delay Product, BDP）来调整发送速率，实现了高吞吐量和低延时。因此，BBR 算法被认为开创了拥塞控制的新纪元。

BBR 算法现已大规模应用于 Google B4 网络与服务器设施。

3.2 原理解析

BBR 由“初始”，“排空”，“探测带宽”与“探测 RTT”四个阶段组成，如图 4 所示。BBR 在这四个阶段通过测量 RTT 和传输速度对瓶颈带宽和传播时延进行交替性的估计，并以此计算使网络链路处于最优操作点的 BDP[6]。

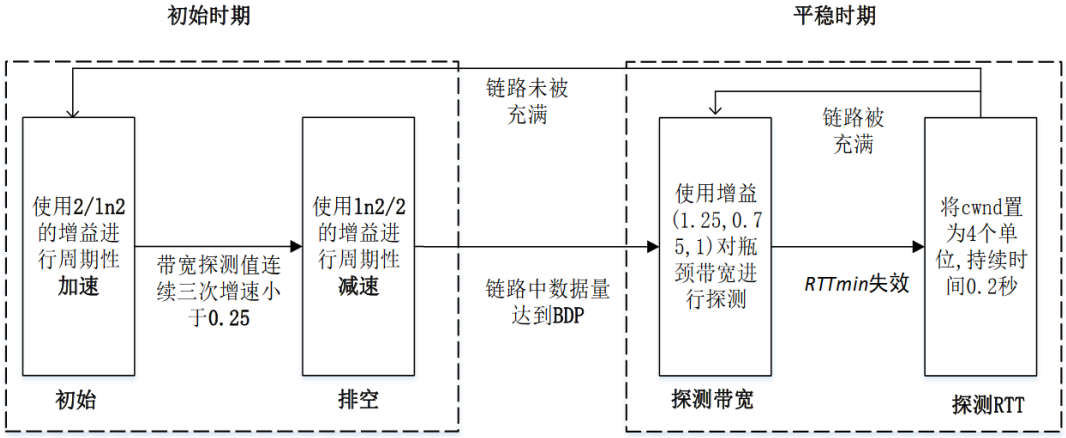

图 4　**BBR** 的状态机

3.2.1 初始阶段（Startup）

在 TCP 连接刚建立时，以增益为 2/ln2 的增长速度加速发送数据，直到探测到的最大带宽在连续 3 个 RTT 内不再增加,通过计算得到最优操作点时的 BDP。这种调节机制使得 BBR 能快速地探测网络带宽[6]。

3.2.2 排空阶段（Drain）

在排空阶段，在该阶段 BBR 以初始阶段速率的倒数发送数据包，直到网络中的数据包数量小于等于 BDP。其目的在于排空初始阶段超发的数据包[7]。

3.2.3 探测带宽阶段（Probe_BW）

探测带宽阶段是一个稳定阶段，也是 BBR 的主要阶段。这一阶段 BBR 通过累乘增益系数，使发送速率主动适应瓶颈带宽的变化。

3.2.4 探测 RTT 阶段（Probe_RTT）

当最小延迟在设定时间内未更新时，减少数据包发送量，以尝试检测更小的最小 RTT，然后在检测完成后根据最新延迟数据确定进入的下一个阶段[8]（初始或探测带宽阶段）。

3.3 参考代码

BBR 算法核心伪代码由两部分组成[4]。

3.3.1 接收 ACK

根据每个 ACK （acknowledgement，响应应答）更新 RTprop 和 BtlBw 估计值。

```
function onAck(packet)
  rtt = now - packet.sendtime
  update_min_filter(RTpropFilter, rtt)
  delivered += packet.size
  delivered_time = now
  deliveryRate = (delivered - packet.delivered) / (delivered_time -
packet.delivered_time)
  if (deliveryRate > BtlBwFilter.currentMax || ! packet.app_limited)
    update_max_filter(BtlBwFilter, deliveryRate)
  if (app_limited_until > 0)
    app_limited_until = app_limited_until - packet.size
```

3.3.2 发送数据

对每个数据包发送速率进行调整，使数据包到达速率与瓶颈链路的出发速率相匹配。

```
function send(packet)
  bdp = BtlBwFilter.currentMax × RTpropFilter.currentMin
  if (inflight >= cwnd_gain × bdp)
    // wait for ack or retransmission timeout
    return
  if (now >= nextSendTime)
    packet = nextPacketToSend()
    if (! packet)
      app_limited_until = inflight
      return
    packet.app_limited = (app_limited_until > 0)
    packet.sendtime = now
    packet.delivered = delivered
    packet.delivered_time = delivered_time
    ship(packet)
    nextSendTime = now + packet.size / (pacing_gain × BtlBwFilter.currentMax)
  timerCallbackAt(send, nextSendTime)
```

3.4 算法性能

3.4.1 算法优势

BBR 算法不将丢包视为拥塞，所以在丢包率较高的网络中，BBR 依然有极高的吞吐量。如图 5 所示。在 1% 丢包率的网络环境下，CUBIC 的吞吐量已经降低 90% 以上，而 BBR 的吞吐量几乎没有受到影响，当丢包率大于 5% 时，BBR 的吞吐量才出现明显的下降，但是当丢包率小于 10^{-5} 时，BBR 吞吐量不如 CUBIC[9]。

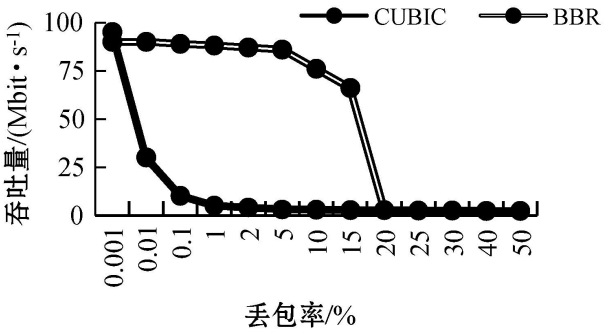

图 **5**　**BBR** 与 **CUBIC** 在不同丢包率情况下吞吐量

3.4.2 算法缺陷

一些评估报告指出，BBR 算法仍存在着一些问题。比如 BBR 与 Reno/CUBIC 共享瓶颈时存在不公平性，攻击性太强；在浅缓冲区中丢包率高；具有不同 RTT 的数据流之间存在带宽不公平性等[10]。

2019 年，BBR 发布了新的 alpha 版本，即版本 2（BBRv2），致力于解决这些局限性。BBRv2 使用两个估算值来确定连接的发送速率：瓶颈带宽和连接的 RTT，并采用带宽探测，以便更好地与 Reno/CUBIC 共存。BBRv2 还采用了显式拥塞通知 (ECN)，并能更好地估计丢包率，以确定发送速率[11]。

4 总结

计算机网络依赖于数量庞大的基础设施与传统协议，在给定框架内提出新的思路，是拥塞控制算法的长远任务。BBR 算法由于其优越性倍受青睐，正在取代传统的 TCP 拥塞控制算法，例如 Linux 内核从 4.9 版本开始支持 BBR 算法。

BBR 算法（v2，v3）已大规模应用于 Google B4 网络，并且与新传输协议 QUIC 协议相互配合，实现了更好的传输性能和使用体验，具有巨大的发展潜力。但是对于其存在的具体缺陷，还需要进一步研究和改进，这也是 BBR 尚未在家用设备上普及的一个原因。

参考文献

[1] 曹涛涛. 拥塞控制算法的性能评估及公平性分析[D]. 2017.

[2] NETLAB C. Communication Networks[EB/OL]. (2023). http://netlab.caltech.edu/research/communication-networks/.

[3] FISHER B, MICHELONI G, BEMMEL J van, 等. Introduction[EB/OL]. (2023). https://tcpcc.systemsapproach.org/intro.html.

[4] CARDWELL N, CHENG Y, GUNN C S, 等. BBR: Congestion-Based Congestion Control[J/OL]. ACM Queue, 2016: 20-53. http://queue.acm.org/detail.cfm?id=3022184.

[5] MAMATAS L, HARKS T, TSAOUSSIDIS V. Approaches to Congestion Control in Packet Networks[J/OL]. Journal of Internet Engineering, 2007, 1(1). https://web.archive.org/web/20140221123729/http://utopia.duth.gr/~emamatas/jie2007.pdf.

[6] 杨明. 网络拥塞控制算法 BBR 的公平性研究与改进[D/OL]. 2022. DOI:10.27157/d.cnki.ghzku.2020.002731.

[7] 王志远. BBR 拥塞控制算法延迟及带宽探测优化[J/OL]. 计算机与现代化, 2022, 0(113–120). DOI:10.3969/j.issn.1006-2475.2022.10.019.

[8] 马力文. BBR 单边拥塞控制算法传输性能优化[D/OL]. 2023. DOI:10.27251/d.cnki.gnjdc.2022.001243.

[9] 王云飞, 黄勇, 朱珊, 等. 改进 BBR 算法在卫星网 TCP 加速网关中的应用研究[J/OL]. 计算机应用与软件, 2023, 40(1): 163-166. DOI:10.3969/j.issn.1000-386x.2023.01.026.

[10] 潘婉苏. TCP-BBR 拥塞控制算法的公平性优化研究[D/OL]. 2023. DOI:10.27517/d.cnki.gzkju.2022.000281.

[11] TIERNEY B, DART E, KISSEL E, 等. Exploring the BBRv2 Congestion Control Algorithm for use on Data Transfer Nodes[D/OL]. 2021. https://fasterdata.es.net/assets/Uploads/INDIS-2021-bbr2.final.pdf.