拥塞控制算法综述

田宝林　韩昊轩

目录

.

创建于: 2020 年 9 月 21 日
更新于: 2020 年 10 月 5 日

1 背景

尽管 TCP 协议中的拥塞控制算法已经发展了几十年，但是互联网中的 TCP 数据包传输效率依然非常的低效。很难有一个拥塞控制算法，能够统一的解决所有性能上的问题，这与它们底层的设计有关：在包的层面硬绑定 (Hardwired Mapping)[6] 了对应的拥塞控制算法。例如，传统的拥塞控制协议是基于数据链路是否丢包进行反应的。一旦产生丢包，那么拥塞控制窗口就会减半。丢包往往可能有两方面的原因 [15]：

- 数据链路中真的产生了缓冲队列的拥塞，因此需要减小拥塞控制窗口。
- 数据链路传输错误，导致数据包丢包。

因传输环境和传输介质的不稳定性，第二种情况是有固定的概率产生。但是这种情况下不需要进行拥塞控制窗口减半的操作，因此降低了 TCP 数据包的传输效率。

随着大规模互联网应用的不断发展，如搜索引擎、视频网站、网络商店，千万级别的数据访问量对数据的访问操作提出了极大的挑战。服务提供商往往有自己的数据中心 (Data Center) 来处理这些海量的请求，并且合理的分发任务进行高效的处理。在数据传输的过程中，也对服务质量 (SLA) 提出了挑战。数据中心数据传输和传统的广域网的数据传输有着非常多不同的特点：

- 随着大规模集成电路技术的发展，硬件的制造成本越来越低，使得在广域网中的链路设备缓存大小越来越大，链路呈现长肥管道状 (long haul network)[20]。但是在数据中心中，并不会为了追求低丢包率而设置较大的缓存。相反，为了能够及时的相应的数据请求，往往设置非常小的缓冲区大小。若有较多的数据请求包到来，并且占满了小缓冲区的时候，那么就会直接进行丢包。
- 数据中心的时延往往是非常低的，时延的数量级大约是 μs 级别，因此基于时延的普通拥塞控制协议是很难在数据中心中起来作用的 [23]。

因此，将传统的拥塞控制算法应用在数据中心会产生很多的问题，如 Vegas[3], FAST[18] 拥塞控制算法是基于链路的时延的，数据中心的数据发送时延大约在 10 微秒左右，因此传统的拥塞控制算法很难准确的测量瓶颈链路的时延，造成数据传输的低效。因此在 2010 年左右，Microsoft 首先推出了 DCTCP[1] 算法，开始了数据中心拥塞控制的分支。

除了传统广域网和数据中心拥塞控制算法的更新和发展，在广域网的拥塞控制算法中细分出了更多的研究门类。目前有下面的几个类别：

1. Reno, Cubic 等拥塞控制算法先填满整个链路，产生丢包，然后做降速，进行周期调整。

2. 不使用丢包率作为拥塞调整的信号，而是使用时延信号进行控制。

3. 特定领域的拥塞控制算法，不再追求一般性。

4. 基于学习算法的在线/离线拥塞控制协议。

其中传统一类的基于规则控制思想的拥塞控制算法；还有一类使用目前非常热门的机器学习的相关知识，先给定相应的优化模型和优化目标，根据网络中收集到的数据，动态的调整整个拥塞算法的控制参数。

由于网络环境的多变性，如蜂窝网络、无线网络 (Adaptive Congestion Control for Unpredictable Cellular Networks)、网页应用、视频传输、有线网络、卫星网络，很难有一个确定统一的算法，能够较为有效的处理所有的网络情况，出现越来越多具体的拥塞控制算法，在特定的领域能够取得较为高效的传输效率。

设计一个新的拥塞控制算法时，需要确定是在应用层做还是在内核层面进行，并且需要和其他的拥塞控制算法进行对比。但是由于测试环境的多样性，太多的研究实践花在复现他人的实验环境和结果，造成了精力的浪费。其次，一个实体的云服务器通过虚拟化的技术来分割成多个虚拟机，从而能够以更细粒度的时间和空间管理有限的计算资源。但是，当租赁用户需要更换拥塞控制协议，他们是没有权限 [9] 来进行这样的操作的。有多种研究对新型拥塞控制算法的易用性平台和底层平台 [22] 支持做了很多的努力。希望能够提供一个通用的平台，使用统一的测试环境，从而尽可能减小复现的重复劳作，提高设计和测试的效率。目前主流通用测试平台有 CCP[11] 和 Pantheon[7]。

本 Survey 主要探索目前主流的拥塞控制算法，给出背后相同的设计思路，给出它们使用的优点和局限性，从而能够更好理解拥塞控制算法根本局限性。

2 各类网络的特点

由于网络系统环境的多变性，我们首先需要了解一些网络场景中的特点，从而能够帮助我们更好的理解需要解决的场景问题，同时能够更好的理解每种特定网络的拥塞控制协议背后设计的思想，最后能够方便我们对相应的网络进行模拟 (Simulation) 和仿真 (Emulation)。

2.1 广域网

- 在建立网络基础设备的初期，往往硬件的缓冲区大小是非常小的。随着摩尔定律的发展，硬件的容量已经不再是问题，因此目前的网络链路的特点为长肥管道状 (Long Haul Network)。[20]
- 数据流速度变化非常的大，网络变化大。例如在生活中，有时候在视频网站上看视频可能会无常的出现视频卡顿的现象。

2.2 数据中心

- Incast: 大量的小数据请求同时从机架 (Racks) 发往中心处理点 (Aggregator)，然后填满了缓冲区，使得丢包，发出重传信号，超时重传时间 (Retransmission Timeout) 在客户端一般是 100ms 左右，而数据中心的时延一般小于 1ms，这样就导致数据中心会等待客户端的重传。降低了 90%Throughput[1]。
- 带宽为 1-40Gbps，时延非常的低，微秒级别。

2.3 移动网络

- 短时间内网速变化非常大
- 自身导致的排队时延
- 与拥塞控制无关的丢包

2.4 卫星网络

- 较高的随机丢包率
- 高时延

3 优化目标

衡量网络好坏往往有非常多的指标，不同的网络系统往往追求的指标是不相同的。如股票交易系统、实时网络游戏的网络对时延有着非常高的要求；网络视频、网络语音往往对网络的丢包率有较高的要求。这些系统背后往往会有某个具体的拥塞控制协议来控制管理的。论文《An Experimental Study of the Learnability of Congestion Control》[14] 描述了一个在上帝视角 (Omniscient) 的全能型的拥塞控制算法，在所有的性能方面都是最优的。下面将列出几个通用的测量指标，方便通过这些指标的变化来评价具体拥塞控制算法的好坏，同时也能够更好的知道网络中的优化目标。

3.1 高带宽

一些长连接 (Long Term Connection) 追求高吞吐，是主要的优化的目标的应用场景。一些短连接对低时延有较高的要求。

3.2 低时延

在广域网中，由于现在网络互联设备不断的更新，内存越来越廉价。因此，广域网中呈现长肥管道的现象越来约明显。因此时延很难控制。

在数据中心内部，一般的做法就是减小缓冲区的大小，从而能够减小排队时延。

3.3 丢包率

比较传统的拥塞控制算法是基于丢包这个指标进行拥塞控制的，使得在没有发生丢包动作的时候，算法会拼命的往链路中发送数据包，来填满整个链路，因此这个策略往往丢包率是比较低的。在进行控制的过程中，丢包率也是一个非常重要的一项指标。

3.4 竞争性

不同种类 TCP 数据流 [17] 公平性是非常重要的，虽然每种拥塞控制算法优化指标有不同，但是在竞争性问题上还是必须达成一致的。由于互联网底层的发展还是非常的缓慢的，其中保存着很多约定俗成的规则。若轻易的破坏数据链路公平竞争的性质，那么会产生恶意竞争的情态，造成互联网的混乱。

设有 n 个数据流，每个数据流的流速为 f_i，评估指标为 Jain index:

$$index(f_1, f_2, ..., f_n) = \frac{(\sum_{i=1}^{n} f_i)^2}{n * \sum_{i=1}^{n} f_i^2}$$

该数值的取值范围为 0-1，越接近 1 表示越公平。

3.4.1 竞争性问题的探讨

关于网络链路公平竞争是一个非常重要的问题，因为网络设备的更新往往是非常慢的，中间链路存在着很多的较陈旧拥塞控制算法。若此刻新的拥塞控制算法加入网络，导致数据链路中的旧的拥塞控制算法无法公平的占有应有的带宽资源。若对这种情况放任不管，势必会导致拥塞控制算法的恶性发展，甚至会导致整个网络大面积拥塞而最终谁都得不到好处。

正是对公平性的考虑，BBR 拥塞控制算法对链路带宽资源的抢占较为的激进，有研究表示 [14]，在 16 条 Cubic 控制的网络终端和一条 BBR 控制的网络终端，它们共享一个带宽链路时，一条 BBR 控制的链路会占用 50% 的带宽资源。随后的迭代版本 v2[13] 中，BBR 取消了较为激进的拥塞控制策略，一定程度上缓解了公平性的问题。

然而，这样的共识却不一定是强制的。明面上，所有经过大规模部署的拥塞控制算法必须遵循这样的准则。但是存在着非常多的” 改进版” 的拥塞控制算法，它们往往会疯狂的抢占带宽资源，而不会在意整个链路的公平性，如 BBR 魔改版，锐速等等 [5] 广为流传的拥塞控制协议应用。

4 主要拥塞控制算法

通过调研，发现拥塞控制算法使用的信号各不相同。及时是使用了相同的信号量，采集的方式，处理的方式也千差万别，这就造成了拥塞控制算法的多样性。主要的拥塞信号包括时延和丢包率 [19]。下面我们将通过具体的算法，看看这些信号是如何被利用的。

4.1 广域网一般拥塞控制算法

4.1.1 Reno

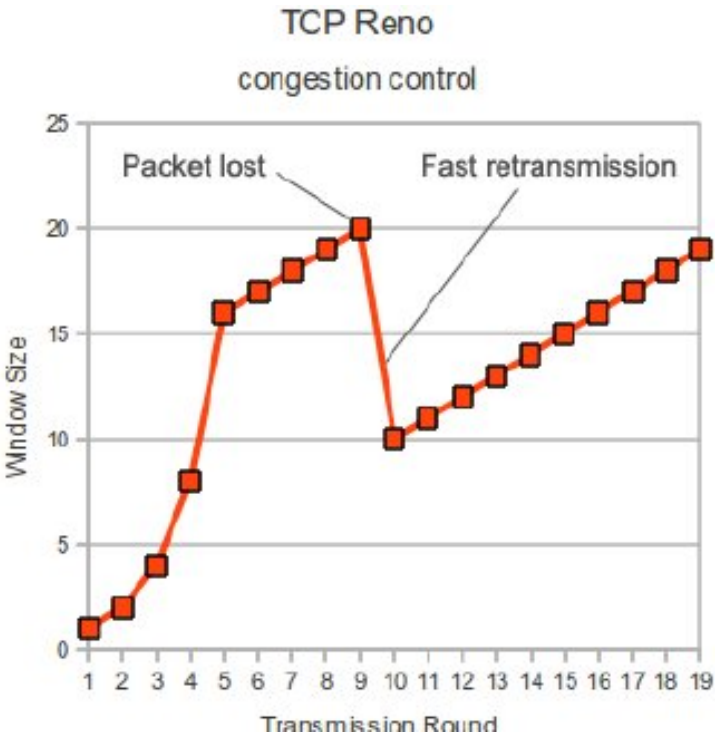


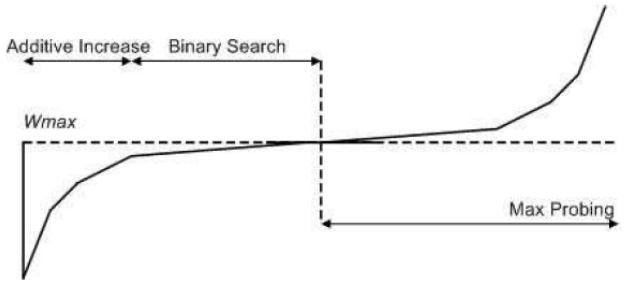
图 1 reno 控制拥塞控制窗口变化

结合图1, 可以将整个拥塞控制的流程分成四个部分，主要是下面的步骤：

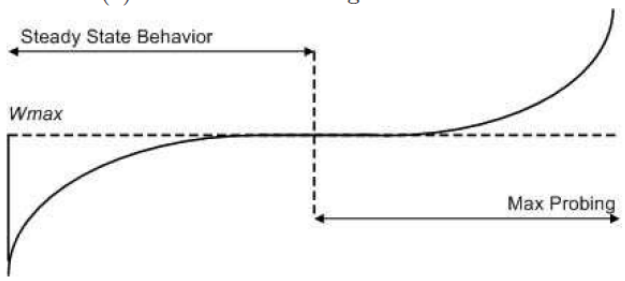
1. 慢启动：当不发生丢包的时候，每一个 RTT 后 cwnd 翻倍
2. 当目前拥塞控制窗口的大小超过原来预设的定值的时候，开始每一个 RTT，cwnd+1，并且更新阈值的大小
3. 快重传：当产生丢包后，拥塞控制窗口的大小立刻减半。
4. 若没有丢包，每一个 RTT 后拥塞控制窗口的大小 +1

4.1.2 Cubic

Cubic 算法 [8] 是对二分法的一个优化，目标是能够拟合出一个合适的三次曲线如图2所示，使得在探测拥塞控制窗口的大小的时候能够减小丢包的数量，从而能够减小对中间链路缓冲区的冲击，同时该方法在离阈值较远的时候，也能很快的收敛到阈值的附近。相当于是利用一些已知的信息，进行曲线的拟合与逼近。



(a) BIC-TCP window growth function.



(b) CUBIC window growth function.

图 2 cubic 拥塞控制窗口增长曲线

4.1.3 PCC

PCC[6] 是一类学习类算法，其选择优化的目标：

1. Throughput: 越高越好
2. Loss: 没有因拥塞控制而产生丢包
3. Latency: 越低越好

其背后的思想很简单：就是以观察到的性能来评判速率变化的方向。在整个控制算法流程中要求输入一组参数，能够实时的调整整个拥塞控制算法协议栈中的各种参数，能够达到最大的利用率。

$$u(x_i, \frac{d(RTT_i)}{dT}, L_i) = x_i^t - bx_i\frac{d(RTT_i)}{dT} - cx_iL_i$$

其中 b, c 都是需要预先确认的常数。在本篇论文中，b=900, c = 11.35。其中 c 阈值越大，那么能够容忍的丢包率越大。b 值的设置与多流的竞争有关系，文中目标 1000 个用户在 1000Mbps 的链路中竞争带宽的时候不会产生较大的抖动 (Inflation)。

4.1.4 BBR

BBR[4] 拥塞控制算法如图3所示。一个循环的周期为 8RTT，其中包含探测期 (ProbeBW)，排空期 (Drain)，稳定期 (Steady State), 前两个状态分别会持续 1RTT，后面的稳定期会持续 6RTT。探测期会主动的多发包，排空期会主动的减少发包的数量，在这个时间段内会探测整个链路的链路的带宽。大约每 10s，网络链路就会对最小 RTT 进行更新。

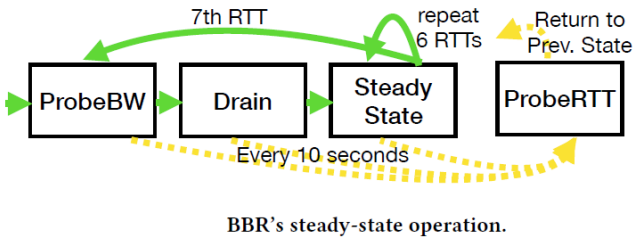


图 3 BBR 算法控制流程

BBR 算法不依照丢包率为指标进行探测。如图4所示，CUBIC 一类以丢包为拥塞控制信号进行控制的算法，实际上在黄实线和红虚线交点处进行不断的探测，并且在周围进行动态的工作。

但是这样的问题是会主动的填满链路中的缓冲区，从而造成链路的拥塞，甚至会产生 Bufferbloat 的问题。BBR 算法主要探测的点为蓝实线和黄实线相交的地方。这样不会主动的去抢占链路中的缓冲区的大小。

简单来讲，就是通过动态的测量瓶颈链路 (Bottleneck Network) 中的瓶颈带宽 ($BBR.BtlBw$) 和检测窗口内的瓶颈链路的传输时延 (BBR.RTProp)，从而两者的乘积产生 BDP(Bandwidth Delay Product)。从而去影响整个 BBR Engine 中的三个参数: $pacingrate$, $sendquantum$ 和 $cwnd$。

4.1.5 Copa

学习类的算法在多变的网络环境下性能比较的好，但是不具有解释性 [6]，能够合理的解释目标函数中的常数的含义往往是非常困难的。

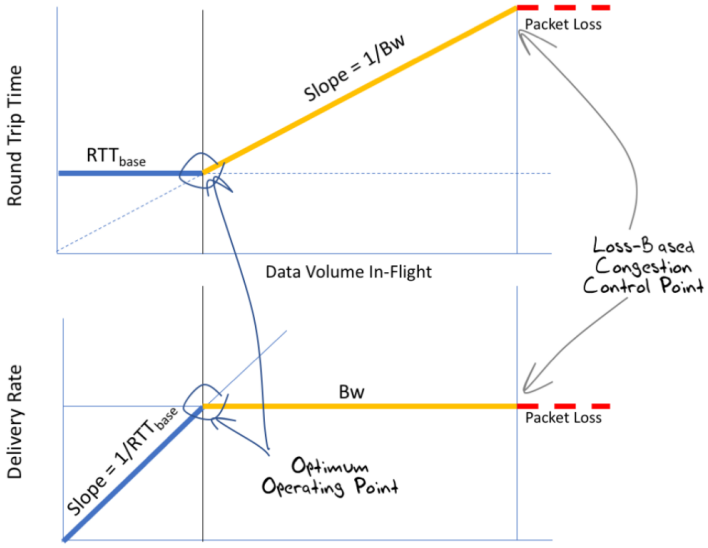


图 4 BBR、Cubic 探测点的区别

Copa[2] 的目标就是在提高可用性的同时，具备很好的解释性。

目标：高速率，低时延，公平竞争。看似前两者是相矛盾的，实际上是 Copa 不会主动的抢占链路中的 Buffer 的大小。但是若链路中有了新的数据流的加入，那么 Copa 会开启竞争模式（开启该模式的指标为 RTT 的抖动大小），从而抛弃低时延的目标，保证传输的速率。

一开始，我们定义一个目标函数：

$$U = \log \lambda - \delta \log d$$

其中 λ 是平均流量，δ 是一个权重系数，d 表示包的时延。

若要使得 U 最大，那么平均流量 λ:

$$\lambda_t = \frac{1}{\delta d_q}$$

$$d_q = RTT_{standing} - RTT_{min}$$

$RTT_{standing}$ 表示在 $\tau = \frac{sRTT}{2}$ 采样窗口求得，RTT_{min} 是在较长一段时间内测得的，一般是 10s 左右。

每收到一个 ACK，就进行下面的更新：

- 若瞬时速率 $\lambda = \frac{cwnd}{RTT_{standing}} \leq \lambda_t$，那么 $cwnd = cwnd + \frac{v}{\delta cwnd}$
- 否则 $cwnd = cwnd - \frac{v}{\delta cwnd}$

其中 v 是一个参数，相当于是包增减的变化快慢的常数，初始化设置为 1。可以得到下面直观的理解：每过一个 RTT，$cwnd$ 大约增加 $\frac{v}{\delta}$。

若要使得拥塞控制窗口进行快速的收敛，那么如果前一个状态为 $cwnd$ 增加，现状态 $cwnd$ 也是增加的，那么 $v = 2 * v$，并且进行拥塞窗口的更新。图5是整个拥塞控制算法的流程，以一个数据流为例，当多流时情况类似。

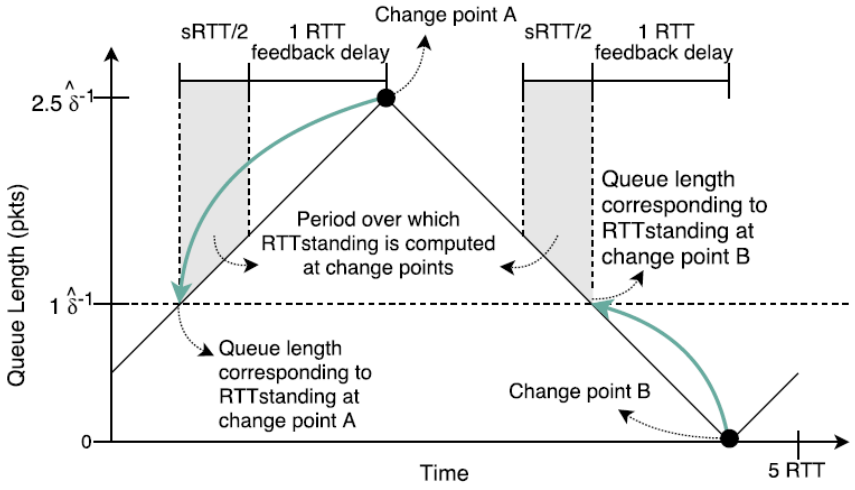


图 5 copa 变化情况

Copa 是一个周期性进行探测的过程，每一个周期为 5RTT 如图5所示。

为什么是 5RTT 而不是 2RTT 呢？首先根据纳什均衡的原理，作者给出了一套理论，当队列中排队的数据包达到 δ^{-1} 时达到竞争均衡。此后进行 $RTT_{standing}$ 的探测，大约持续的时间为 $\frac{RTT}{2}$ 的时间，因为数据包传输需要 1RTT，因此只有达到 Change Point A 时得到 $RTT_{standing}$。反过来的状态类似，直到排空队列中的数据包为止。然后进行周期性的测量与控制。

4.2 数据中心拥塞控制算法

4.2.1 DCTCP

DCTCP[1] 需要在发送端、接收端、交换机中同时进行布置。交换机中设置有常数 K，一旦队列中数据包的数量超过这个常数，那么交换机就会在数据包中设置 CE(Congestion Expericed) Checkpoint 的标志位。

当接收端收到标记的数据包时，本应该立刻发送带有 ECN-echo 标记位的 ACK，但是由于 ACK Compression 的应用，于是又使用了一个如图6所示的 2 State Machine 状态机，从而能够维持 Delayed ACK 的性质。

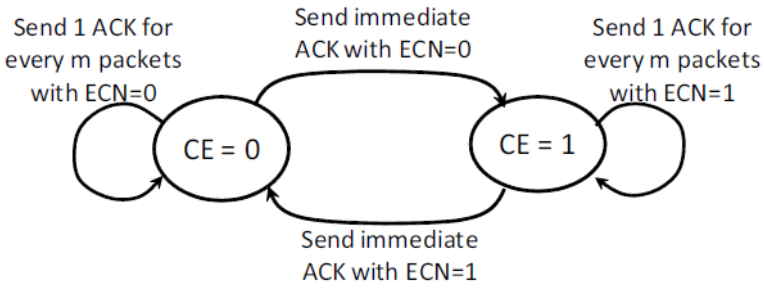


图 6 接收端的状态机

发送端开始统计有多少的数据包被标有 CE Checkpoint，并且维护下面的式子：

$$\alpha = (1-g)\alpha + gF$$

F 表示的是在一个时间窗口内收集到的标记有 CE Checkpoint 的数量，g 是一个权重的系数，α 表示队列中遇到拥塞的概率的大小，$\alpha \in [0,1]$。通过更新 α，从而更进一步的维护 $cwnd = cwnd * (1 - \frac{\alpha}{2})$

当 α 很小的时候，cwnd 保持一个叫小的变化；当 α 接近 1 的时候，cwnd 以指数级别的速率开始减小。

4.2.2 TIMELY

随着硬件路由器的发展，RTT 精度已经达到了毫秒级别的精度。同时为了避免 ACK Compression 等问题，TIMELY[10] 使用的指标并不是单一的 RTT 指标，而是 RTT 梯度。

5 拥塞控制算法参数的其他应用

拥塞控制算法利用数据包携带的若干信息，对复杂多变的网路链路进行主动探测。因此，目前有一些研究结合拥塞控制算法，利用其中的一些参数对网络进行评估，从而能够给产品用户最佳的网络产品使用体验。

5.1 视频画质选择

在传输视频前，视频网站往往会发送少量的数据包进行探测，试图找到最佳的画质[21]，使得用户在享受视频服务的同时，不会出现卡顿的情况，提高了用户的体验。具体的原理时通过少量的数据包的传输，可以准确知道网络速率。从而能够花很小的开销，动态的调整传输视频的画质。

5.2 网络测速

传统的网络测速是使用 UDP 数据包充满 (Saturate) 整个数据链路，不管丢包的数量，从单位时间内发送的数据包的数量来推测整个链路的传输速率。然而这种方法首先没有考虑链路中是否拥塞的情况，即使数据包充满了整个链路，但是测量的结果可能依旧会偏低，这是网络提供商不希望用户所看到的。

若使用一些带有速率探测的拥塞控制协议，如 BBR，首先可以使用拥塞控制算法探测链路是否拥塞，如果拥塞的话可能提早的告诉用户，能够提供更准确的测速不准的原因[16]。其次，基于拥塞控制的网络链路测速不用充满整个链路，从而能够减少数据包的发送，较好的节省了数据流量。

6 Testbed 设计

适用于拥塞控制算法的 Testbed 曾经全是依靠开发者自行针对目标网络系统开发的测试应用，近年来出现了通用的 Testbed 平台。

6.1 Pantheon

Pantheon 是一个用于研究和评估端到端的网络系统的分布式协作平台，特别针对拥塞控制方案、传输协议以及网络仿真器。旨在解决大型网络系统的拥塞控制测试需要开发人员单独开发拥塞控制算法和传输协议的测试平台的问题。

Pantheon 生成网络仿真器，网络仿真器经过校准后匹配到真实的网络路径，对仿真器分级的指标是描述一组算法的端到端性能，例如吞吐量、延迟和丢包率，通过在仿真网络上运行算法，根据端到端的性能信息与真实网络中的相同性能统计信息的路径进行匹配。

Pantheon 被用于了 Vivace 和 Copa 算法等的测试工作。一系列的应用也证实了 Pantheon 既可以将完整的算法部署并测试，也可以进行自动部署和原型测试。通过测量多种传输协议和拥堵控制算法，Pantheon 为研究和改进它们的性能提供了一个训练场所。此外，通过生成匹配现实路径的校准仿真器，Pantheon 使研究人员能够重复和准确地测量协议及算法 [22]。

6.2 CCP

CCP 是一个在数据路径之外进行复杂的拥塞控制算法的测试与评估的系统。数据路径指代数据包在网络上被发送时经由的路由器和交换机的顺序。不同于以往的将拥塞控制算法编译到内核中，CCP 在应用层进行拥塞控制，提出了利用数据包的数据路径以及在数据路径上传输时的拥塞信号（包括 RTT，接收，丢失等）的信息，在一个数据路径之外的模块上进行拥塞控制，使数据路径按照指定的拥塞控制策略传输数据，进行拥塞控制算法的测试。

CCP 使得测试人员只需要编译一次拥塞控制算法，即可在所有支持指定接口的数据路径上运行。同时 CCP 与 ACK 时钟解耦，用户可以不受包确认的时间限制，从而开发更加灵活的拥塞控制算法 [12]。

7 个人观点

1. 随着互联网的发展，普适的拥塞控制算法越来越难以满足特定领域的网络指标的需求，随着目标和需求的不断的改变，针对某个特定领域的拥塞控制协议将会越来越多。

2. 尽管最近十年来，机器学习不断的发展深入到了各个领域，但是在网络的拥塞控制领域，性能指标可能有时候还不如一些传统的拥塞控制算法。其原因如机器学习的较难的解释性，很难解决网络链路中纷繁复杂的网络状况，不能成为主流的拥塞控制算法。

3. 利用拥塞控制算法中对链路精准估计的参数，也能提高其他应用服务质量。可以说明拥塞控制是系统层较为基础，非常重要的一个控制模块。

8 Misc

1. 基于一个目标函函数的" 学习" 类算法，他们设计的这个函数往往是非常巧妙的。这涉及到另外一个关于设计目标函数的一般准则：当有多个需要考虑的目标量时，怎么将不同量纲的参数统一到一个式子里面。Copa 使用的方法是使常系数也带有单位，这显然是一个不太好的应用；PCC 使用的方法是统一量纲，但是依旧有一个问题，就是每个式子带有的常参数往往很难解释清楚，并且这些常数的量级差距还是很大的，需要有一个自动调节的算法，来达到算法原来设计的目的。

2. Congestion window 在发送端，TCP Sliding Window 在接收端，不要将两者混淆了，并且本文主要的研究对象为 Congestion Window。

参考文献

[1] Mohammad Alizadeh, Albert G. Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). In Shivkumar Kalyanaraman, Venkata N. Padmanabhan, K. K. Ramakrishnan, Rajeev Shorey, and Geoffrey M. Voelker, editors, Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010, pages 63–74. ACM, 2010.

[2] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In Sujata Banerjee and Srinivasan Seshan, editors, 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018, pages 329–342. USENIX Association, 2018.

[3] Lawrence S. Brakmo and Larry L. Peterson. TCP vegas: End to end congestion avoidance on a global internet. IEEE J. Sel. Areas Commun., 13(8):1465–1480, 1995.

[4] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: congestion-based congestion control. ACM Queue, 14(5):20–53, 2016.

[5] Chikage. Linux-netspeed. https://github.com/chiakge/Linux-NetSpeed.

[6] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-learning congestion control. In Sujata Banerjee and Srinivasan Seshan, editors, 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018, pages 343–356. USENIX Association, 2018.

[7] venkatarun95 francisyyan, greghill. pantheon. https://github.com/StanfordSNR/pantheon.

[8] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: A new tcp-friendly high-speed tcp variant. SIGOPS Oper. Syst. Rev., 42(5):64–74, July 2008.

[9] Keqiang He, Eric Rozner, Kanak Agarwal, Yu (Jason) Gu, Wes Felter, John B. Carter, and Aditya Akella. AC/DC TCP: virtual congestion control enforcement for datacenter networks. In Marinho P. Barcellos, Jon Crowcroft, Amin Vahdat, and Sachin Katti, editors, Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016, pages 244–257. ACM, 2016.

[10] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily R. Blem, Hassan M. G. Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. TIMELY: rtt-based congestion control for the datacenter. In Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye, editors, Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015, pages 537–550. ACM, 2015.

[11] Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Prateesh Goyal, Srinivas Narayana, Radhika Mittal, Mohammad Alizadeh, and Hari Balakrishnan. Ccp: Congestion control plane. https://ccp-project.github.io/.

[12] Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Prateesh Goyal, Srinivas Narayana, Radhika Mittal, Mohammad Alizadeh, and Hari Balakrishnan. Restructuring endpoint congestion control. In Sergey Gorinsky and János Tapolcai, editors, Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018, pages 30–43. ACM, 2018.

[13] Soheil Hassas Yeganeh Neal Cardwell. Bbr implementation. https://github.com/google/bbr.

[14] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. An experimental study of the learnability of congestion control. In Fabián E. Bustamante, Y. Charlie Hu, Arvind Krishnamurthy, and Sylvia Ratnasamy, editors, ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014, pages 479–490. ACM, 2014.

[15] Srikanth Sundaresan, Mark Allman, Amogh Dhamdhere, and kc claffy. TCP congestion signatures. In Steve Uhlig and Olaf Maennel, editors, Proceedings of the 2017

Internet Measurement Conference, IMC 2017, London, United Kingdom, November 1-3, 2017, pages 64–77. ACM, 2017.

[16] Srikanth Sundaresan, Xiaohong Deng, Yun Feng, Danny Lee, and Amogh Dhamdhere. Challenges in inferring internet congestion using throughput measurements. In Steve Uhlig and Olaf Maennel, editors, Proceedings of the 2017 Internet Measurement Conference, IMC 2017, London, United Kingdom, November 1-3, 2017, pages 43–56. ACM, 2017.

[17] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. Modeling bbr's interactions with loss-based congestion control. In Proceedings of the Internet Measurement Conference, IMC 2019, Amsterdam, The Netherlands, October 21-23, 2019, pages 137–143. ACM, 2019.

[18] David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. FAST TCP: motivation, architecture, algorithms, performance. IEEE/ACM Trans. Netw., 14(6):1246–1259, 2006.

[19] Wikipedia. Tcp congestion control. https://en.wikipedia.org/wiki/TCP_congestion_control.

[20] wikipedia. Bandwidth delay product. https://en.wikipedia.org/wiki/Bandwidthdelayproduct, 9 2016.

[21] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In Ranjita Bhagwan and George Porter, editors, 17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020, pages 495–511. USENIX Association, 2020.

[22] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In Haryadi S. Gunawi and Benjamin Reed, editors, 2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018, pages 731–743. USENIX Association, 2018.

[23] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. Adaptive congestion control for unpredictable cellular networks. In Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye, editors, Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015, pages 509–522. ACM, 2015.