12 | 如何调整TCP拥塞控制的性能?

2020-05-25 陶辉

系统性能调优必知必会

进入课程 >

讲述：陶辉

时长 12:47　大小 11.72M

你好，我是陶辉。

上一讲我们谈到接收主机的处理能力不足时，是通过滑动窗口来减缓对方的发送速度。这一讲我们来看看，当网络处理能力不足时又该如何优化 TCP 的性能。

如果你阅读过 TCP 协议相关的书籍，一定看到过慢启动、拥塞控制等名词。这些概念似乎离应用开发者很远，然而，如果没有拥塞控制，整个网络将会锁死，所有消息都无法传输。

而且，如果你在开发分布式集群中的高并发服务，理解拥塞控制的工作原理，就可以在内核的 TCP 层，提升所有进程的网络性能。比如，你可能听过，2013 年谷歌把初始拥塞窗口

从 3 个 MSS（最大报文长度）左右提升到 10 个 MSS，将 Web 站点的网络性能提升了 10% 以上，而有些高速 CDN 站点，甚至把初始拥塞窗口提升到 70 个 MSS。

特别是，近年来谷歌提出的 BBR 拥塞控制算法已经应用在高版本的 Linux 内核中，从它在 YouTube 上的应用可以看到，在高性能站点上网络时延有 20% 以上的降低，传输带宽也有提高。

Linux 允许我们调整拥塞控制算法，但是，正确地设置参数，还需要深入理解拥塞控制对 TCP 连接的影响。这一讲我们将沿着网络如何影响发送速度这条线，看看如何调整 Linux 下的拥塞控制参数。

慢启动阶段如何调整初始拥塞窗口？

上一讲谈到，只要接收方的读缓冲区足够大，就可以通过报文中的接收窗口，要求对方更快地发送数据。然而，网络的传输速度是有限的，它会直接丢弃超过其处理能力的报文。而发送方只有在重传定时器超时后，才能发现超发的报文被网络丢弃了，发送速度提不上去。更为糟糕的是，如果网络中的每个连接都按照接收窗口尽可能地发送更多的报文时，就会形成恶性循环，最终超高的网络丢包率会使得每个连接都无法发送数据。

解决这一问题的方案叫做拥塞控制，它包括 4 个阶段，我们首先来看 TCP 连接刚建立时的慢启动阶段。由于 TCP 连接会穿越许多网络，所以最初并不知道网络的传输能力，为了避免发送超过网络负载的报文，TCP 只能先调低发送窗口（关于发送窗口，你可以参考[第 11 讲]），减少飞行中的报文来让发送速度变慢，这也是“慢启动”名字的由来。

让发送速度变慢是通过引入拥塞窗口（全称为 congestion window，缩写为 CWnd，类似地，接收窗口叫做 rwnd，发送窗口叫做 swnd）实现的，它用于避免网络出现拥塞。上一讲我们说过，如果不考虑网络拥塞，发送窗口就等于对方的接收窗口，而考虑了网络拥塞后，发送窗口则应当是拥塞窗口与对方接收窗口的最小值：

```
1  swnd = min(cwnd, rwnd)
```

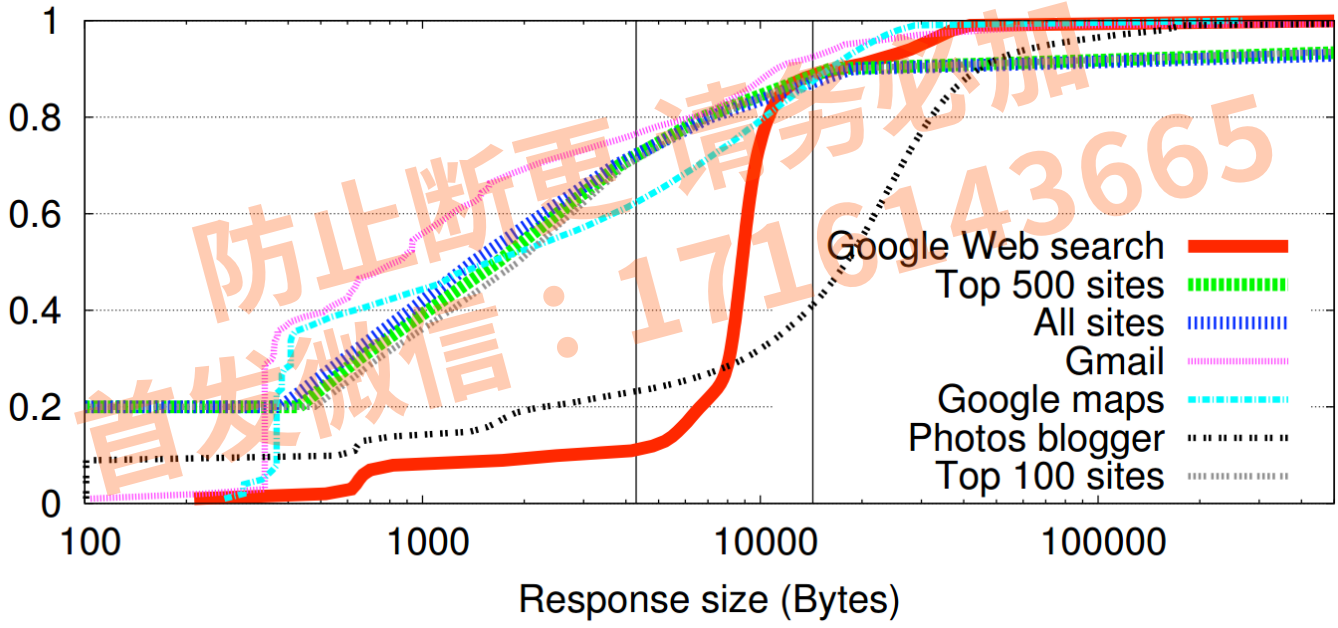
复制代码

这样，发送速度就综合考虑了接收方和网络的处理能力。

虽然窗口的计量单位是字节，但为了方便理解，通常我们用 MSS 作为描述窗口大小的单位，其中 MSS 是 TCP 报文的最大长度。

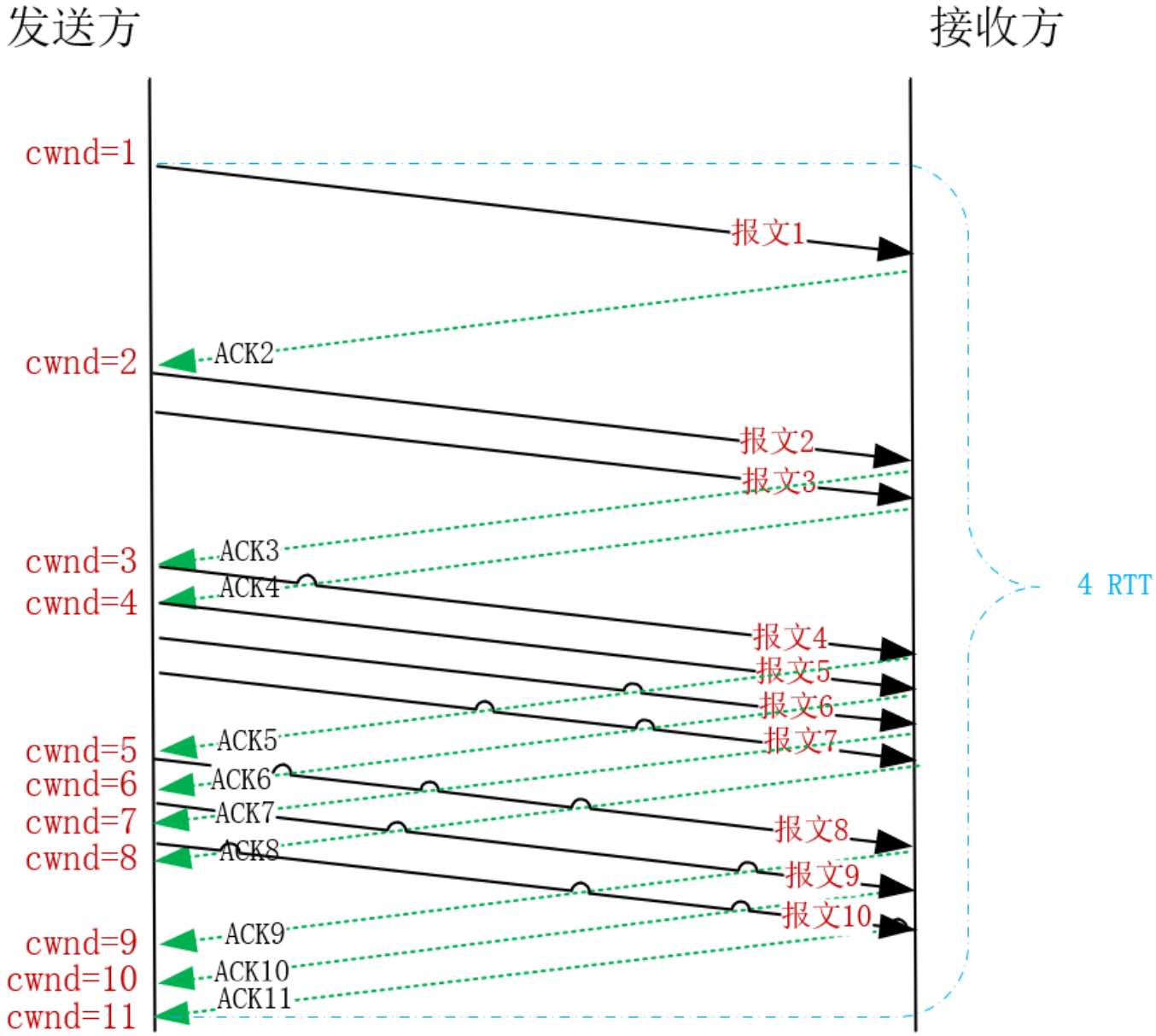
如果初始拥塞窗口只有 1 个 MSS，当 MSS 是 1KB，而 RTT 时延是 100ms 时，发送速度只有 10KB/s。所以，当没有发生拥塞时，拥塞窗口必须快速扩大，才能提高互联网的传输速度。因此，慢启动阶段会以指数级扩大拥塞窗口（扩大规则是这样的：发送方每收到一个 ACK 确认报文，拥塞窗口就增加 1 个 MSS），比如最初的初始拥塞窗口（也称为 initcwnd）是 1 个 MSS，经过 4 个 RTT 就会变成 16 个 MSS。

虽然指数级提升发送速度很快，但互联网中的很多资源体积并不大，多数场景下，在传输速度没有达到最大时，资源就已经下载完了。下图是 2010 年 Google 对 Web 对象大小的 CDF 累积分布统计，大多数对象在 10KB 左右。

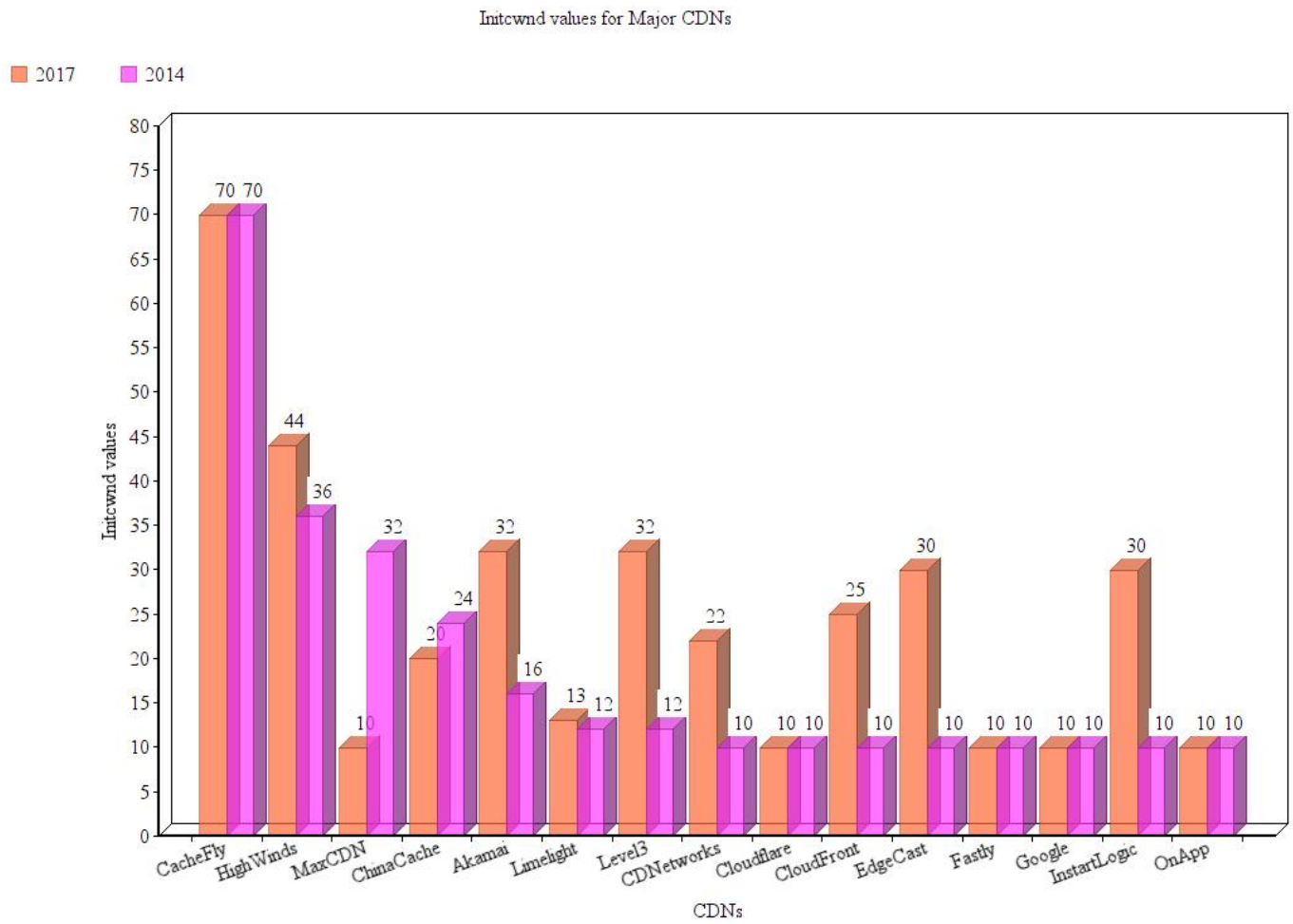


图片来源：《An Argument for Increasing TCP's Initial Contestion Window》

这样，当 MSS 是 1KB 时，多数 HTTP 请求至少包含 10 个报文，即使以指数级增加拥塞窗口，也需要至少 4 个 RTT 才能传输完，参见下图：



因此，2013 年 TCP 的初始拥塞窗口调整到了 10 个 MSS（参见RFC6928），这样 1 个 RTT 内就可以传输 10KB 的请求。然而，如果你需要传输的对象体积更大，BDP 带宽时延积很大时，完全可以继续提高初始拥塞窗口的大小。下图是 2014 年、2017 年全球主要 CDN 厂商初始拥塞窗口的变化，可见，随着网速的增加，初始拥塞窗口也变得更大了。



图片来源：https://blog.imaginea.com/look-at-tcp-initcwnd-cdns/

因此，你可以根据网络状况和传输对象的大小，调整初始拥塞窗口的大小。调整前，先要清楚你的服务器现在的初始拥塞窗口是多大。你可以通过 ss 命令查看当前拥塞窗口：

复制代码

```
# ss -nli|fgrep cwnd
         cubic rto:1000 mss:536 cwnd:10 segs_in:10621866 lastsnd:1716864402 la
```

再通过 ip route change 命令修改初始拥塞窗口：

复制代码

```
# ip route | while read r; do
            ip route change $r initcwnd 10;
        done
```

当然，更大的初始拥塞窗口以及指数级的提速，连接很快就会遭遇网络拥塞，从而导致慢启动阶段的结束。

出现网络拥塞时该怎么办？

以下 3 种场景都会导致慢启动阶段结束：

1. 通过定时器明确探测到了丢包；
2. 拥塞窗口的增长到达了慢启动阈值 ssthresh（全称为 slow start threshold），也就是之前发现网络拥塞时的窗口大小；
3. 接收到重复的 ACK 报文，可能存在丢包。

我们先来看第 1 种场景，在规定时间内没有收到 ACK 报文，这说明报文丢失了，网络出现了严重的拥塞，必须先降低发送速度，再进入拥塞避免阶段。不同的拥塞控制算法降低速度的幅度并不相同，比如 CUBIC 算法会把拥塞窗口降为原先的 0.8 倍（也就是发送速度降到 0.8 倍）。此时，我们知道了多大的窗口会导致拥塞，因此可以把慢启动阈值设为发生拥塞前的窗口大小。

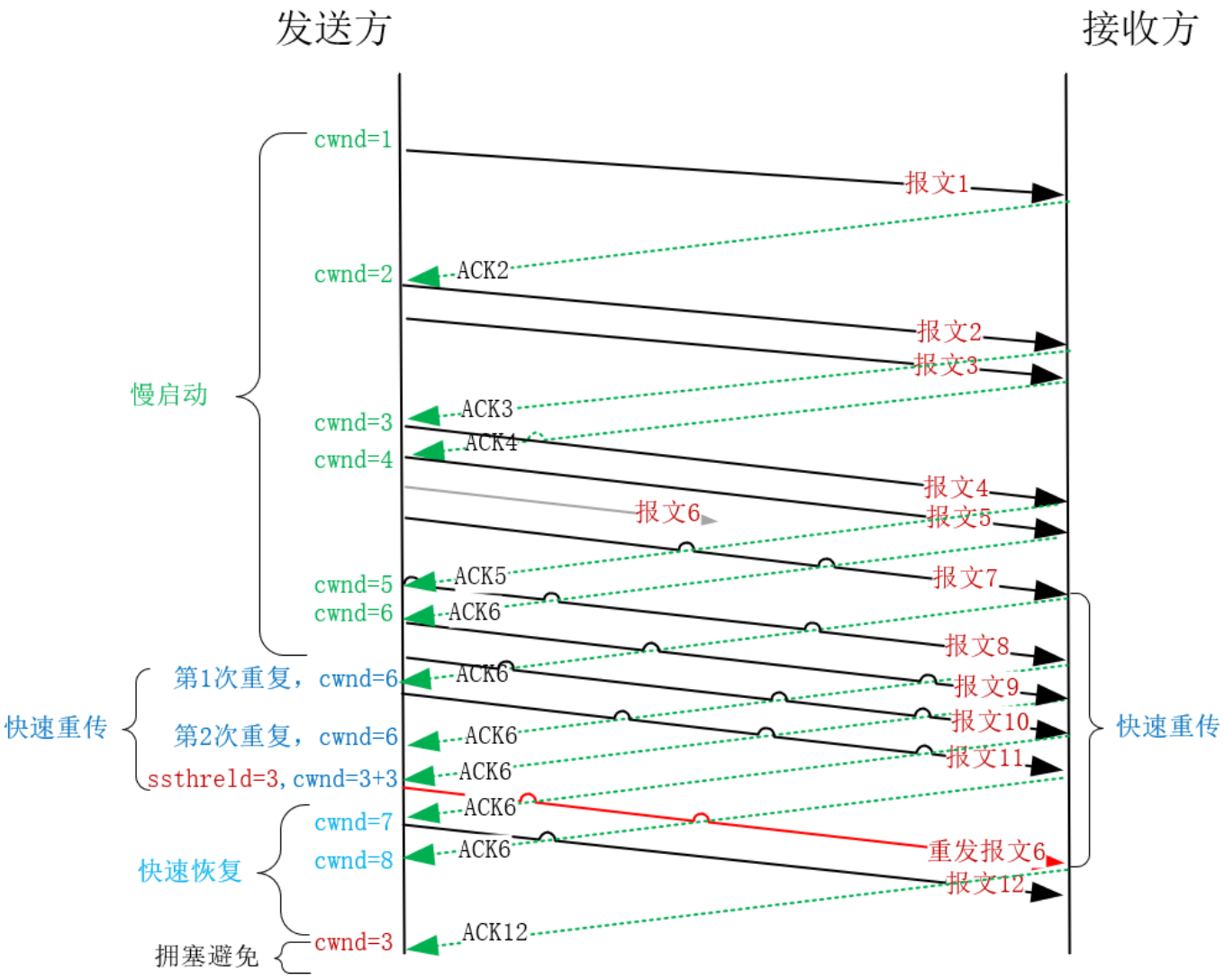
再看第 2 种场景，虽然还没有发生丢包，但发送方已经达到了曾经发生网络拥塞的速度（拥塞窗口达到了慢启动阈值），接下来发生拥塞的概率很高，所以进入**拥塞避免阶段，此时拥塞窗口不能再以指数方式增长，而是要以线性方式增长。**接下来，拥塞窗口会以每个 RTT 增加 1 个 MSS 的方式，代替慢启动阶段每收到 1 个 ACK 就增加 1 个 MSS 的方式。这里可能有同学会有疑问，在第 1 种场景发生前，慢启动阈值是多大呢？事实上，RFC5681 建议最初的慢启动阈值尽可能的大，这样才能在第 1、3 种场景里快速发现网络瓶颈。

第 3 种场景最为复杂。我们知道，TCP 传输的是字节流，而“流”是天然有序的。因此，当接收方收到不连续的报文时，就可能发生报文丢失或者延迟，等待发送方超时重发太花时间了，为了缩短重发时间，**快速重传算法便应运而生。**

当连续收到 3 个重复 ACK 时，发送方便得到了网络发生拥塞的明确信号，通过重复 ACK 报文的序号，我们知道丢失了哪个报文，这样，不等待定时器的触发，立刻重发丢失的报文，可以让发送速度下降得慢一些，这就是快速重传算法。

出现拥塞后，发送方会缩小拥塞窗口，再进入前面提到的拥塞避免阶段，用线性速度慢慢增加拥塞窗口。然而，**为了平滑地降低速度，发送方应当先进入快速恢复阶段，在失序报文到达接收方后，再进入拥塞避免阶段。**

那什么是快速恢复呢？我们不妨把网络看成一个容器（上一讲中说过它可以容纳 BDP 字节的报文），每当接收方从网络中取出一个报文，发送方就可以增加一个报文。当发送方接收到重复 ACK 时，可以推断有失序报文离开了网络，到达了接收方的缓冲区，因此可以再多发送一个报文。如下图所示：



这里你要注意：第 6 个报文在慢启动阶段丢失，接收方收到失序的第 7 个报文会触发快速重传算法，它必须立刻返回 ACK6。而发送方接收到第 1 个重复 ACK6 报文时，就从慢启动进入了快速重传阶段，**此刻的重复 ACK 不会扩大拥塞窗口。**当连续收到 3 个 ACK6 时，发送方会重发报文 6，并把慢启动阈值和拥塞窗口都降到之前的一半：3 个 MSS，再进入快速恢复阶段。按照规则，由于收到 3 个重复 ACK，所以拥塞窗口会增加 3 个 MSS。之后收到的 2 个 ACK，让拥塞窗口增加到了 8 个 MSS，直到收到期待的 ACK12，发送方才会进入拥塞避免阶段。

慢启动、拥塞避免、快速重传、快速恢复，共同构成了拥塞控制算法。Linux 上提供了更改拥塞控制算法的配置，你可以通过 tcp_available_congestion_control 配置查看内核支持

的算法列表：

```
net.ipv4.tcp_available_congestion_control = cubic reno
```

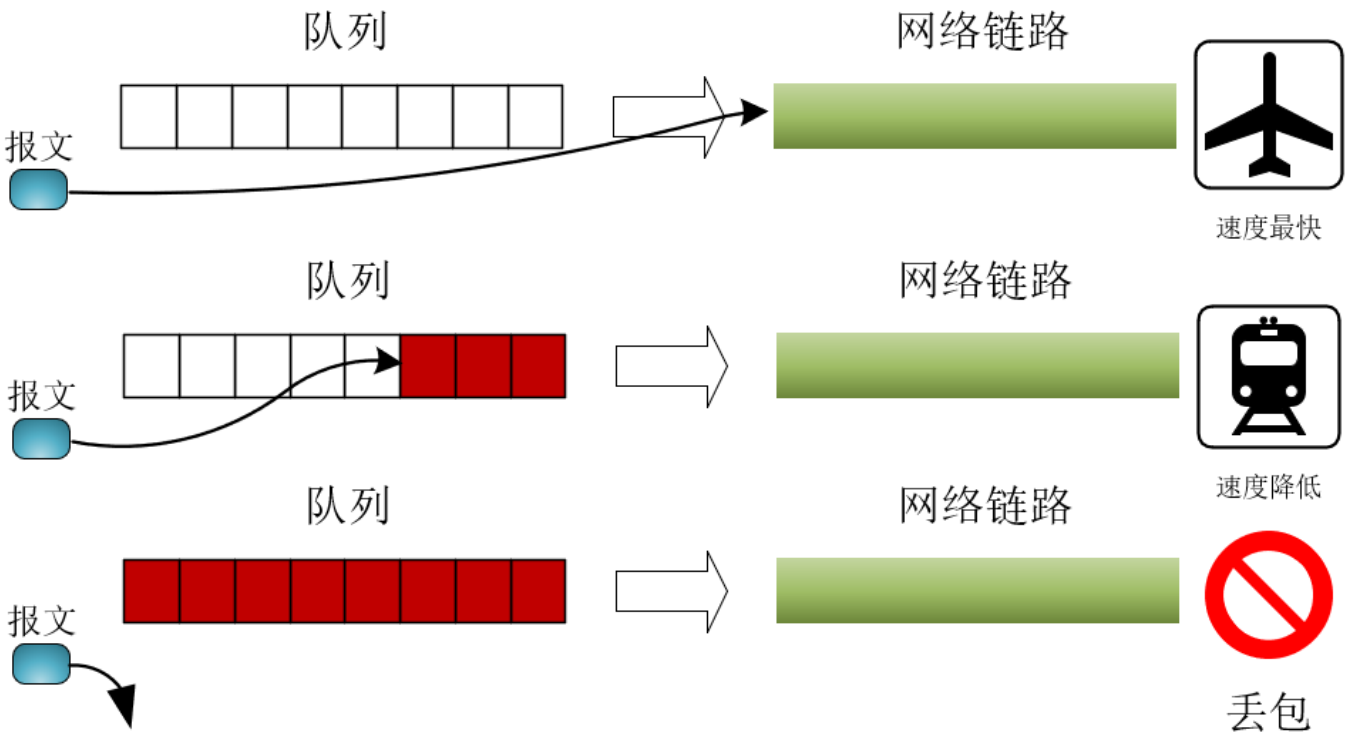
再通过 tcp_congestion_control 配置选择一个具体的拥塞控制算法：

```
net.ipv4.tcp_congestion_control = cubic
```

但有件事你得清楚，拥塞控制是控制网络流量的算法，主机间会互相影响，在生产环境更改之前必须经过完善的测试。

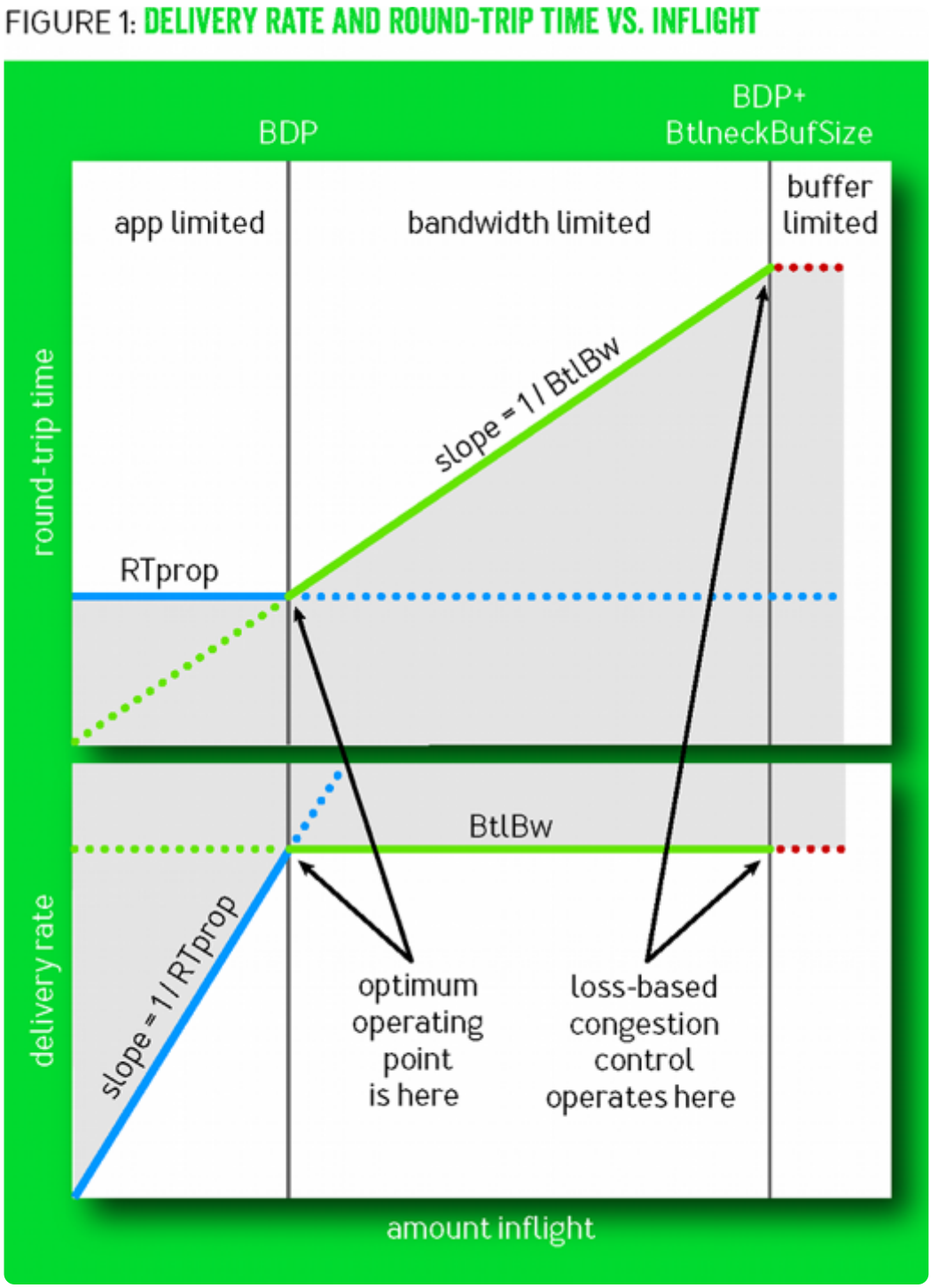
基于测量的拥塞控制算法

上文介绍的是传统拥塞控制算法，它是以丢包作为判断拥塞的依据。然而，网络刚出现拥塞时并不会丢包，而真的出现丢包时，拥塞已经非常严重了。如下图所示，像路由器这样的网络设备，都会有缓冲队列应对突发的、超越处理能力的流量：



当缓冲队列为空时，传输速度最快。一旦队列开始积压，每个报文的传输时间需要增加排队时间，网速就变慢了。而当队列溢出时，才会出现丢包，基于丢包的拥塞控制算法在这个时间点进入拥塞避免阶段，显然太晚了。因为升高的网络时延降低了用户体验，而且从丢包到重发这段时间，带宽也会出现下降。

进行拥塞控制的最佳时间点，是缓冲队列刚出现积压的时刻，**此时，网络时延会增高，但带宽维持不变，这两个数值的变化可以给出明确的拥塞信号**，如下图所示：



图片来源网络：传输速度_RTT与飞行报文的关系

这种以测量带宽、时延来确定拥塞的方法，在丢包率较高的网络中应用效果尤其好。2016 年 Google 推出的 BBR 算法（全称 Bottleneck Bandwidth and Round-trip propagation time），就是测量驱动的拥塞控制算法，它在 YouTube 站点上应用后使得网络时延下降了 20% 以上，传输带宽也有 5% 左右的提升。

当然，测量驱动的拥塞算法并没有那么简单，因为网络会波动，线路也会变化，算法必须及时地响应网络变化，这里不再展开算法细节，你可以在我的这篇博客中找到 BBR 算法更详细的介绍。

Linux 4.9 版本之后都支持 BBR 算法，开启 BBR 算法仍然使用 tcp_congestion_control 配置：

```
1 net.ipv4.tcp_congestion_control=bbr
```

复制代码

小结

我们对这一讲的内容做个小结。

当 TCP 连接建立成功后，拥塞控制算法就会发生作用，首先进入慢启动阶段。决定连接此时网速的是初始拥塞窗口，Linux 上可以通过 route ip change 命令修改它。通常，在带宽时延积较大的网络中，应当调高初始拥塞窗口。

丢包以及重复的 ACK 都是明确的拥塞信号，此时，发送方就会调低拥塞窗口减速，同时修正慢启动阈值。这样，将来再次到达这个速度时，就会自动进入拥塞避免阶段，用线性速度代替慢启动阶段的指数速度提升窗口大小。

当然，重复 ACK 意味着发送方可以提前重发丢失报文，快速重传算法定义了这一行为。同时，为了使得重发报文的过程中，发送速度不至于出现断崖式下降，TCP 又定义了快速恢复算法，发送方在报文重新变得有序后，结束快速恢复进入拥塞避免阶段。

但以丢包作为网络拥塞的信号往往为时已晚，于是以 BBR 算法为代表的测量型拥塞控制算法应运而生。当飞行中报文数量不变，而网络时延升高时，就说明网络中的缓冲队列出现了

积压，这是进行拥塞控制的最好时机。Linux 高版本支持 BBR 算法，你可以通过 tcp_congestion_control 配置更改拥塞控制算法。

思考题

最后，请你思考下，快速恢复阶段的拥塞窗口，在报文变得有序后反而会缩小，这是为什么？欢迎你在留言区与大家一起探讨。

感谢阅读，如果你觉得这节课对你有一些启发，也欢迎把它分享给你的朋友。

上一篇　11 | 如何修改TCP缓冲区才能兼顾并发数量与传输速度?

下一篇　13 | 实战：单机如何实现管理百万主机的心跳服务?

精选留言 (6)

写留言

安排

2020-05-25

毕竟还是发生了丢包或者延迟，所以快速恢复后重新进去拥塞避免。

3

忆水寒
2020-05-26

1、关于课后思考题，快速恢复阶段的拥塞窗口，在报文变得有序后反而会缩小?
我想原因可能是 TCP觉得此时网络轻度拥塞，然后拥塞阈值ssthresh降低为cwnd 的一半，并设置cwnd为ssthresh，然后拥塞窗口再线性增长。

2、本文的一些思考与总结...

展开

1

妥协
2020-05-25

为什么报文5之后的ack都是ack6呀

展开

作者回复: 你好妥协，TCP是有序的字符流，因此接收方收完报文5后，只能接收报文6，但现在却接收到了报文7、8、9、10，此时接收方该怎么办呢?
当然，它可以当做不知道，什么也不做，坐等报文6的到来。报文6什么时候会到呢？RTO时间超时后，发送方会重发报文6，因为发送方一直没收到ACK7!
但是，RTO是很长的时间，接收方直接反复的传递ACK6，这样发送方就能明白，报文6丢了，他可以提前重发报文6. 这叫做快速重传!

1

我来也
2020-05-25

第一次见BBR还是在搭建梯子的过程中.
那时ubuntu默认内核还不支持,需要手动开启.

1

分清云淡
2020-05-25

bbr比较适合高rt场景，对机房内网性能反而更差，慎用

展开

1

唐朝首都
2020-05-25

我理解：快速重传是窗口虽然是8，但是在网络中传输的数据包数量是<=4的，因为另外几个被“8，9，10，11”占了，所以进入拥塞避免阶段之后，继续保持8实际上是可能造成拥塞，不如恢复的已经被证明能够支撑的cwnd，然后再线性增长。

展开