答疑（三）| 第11~15讲思考题答案

2022-03-18 杨胜辉

《网络排查案例课》 课程介绍 >

讲述：杨胜辉
时长 10:01 大小 9.18M

你好，我是胜辉。这节课我们继续来解答前面课程的思考题。

第 11 到 14 讲的内容呢，主要是关于 TCP 传输的，涵盖了各种 TCP 重传、超时、拥塞控制、DDoS 等我们都比较关心的问题，这些知识也可以说是 TCP 体系里比较复杂的部分了。到了第 15 讲，就进入应用层真实案例了，可能离开发和运维开发的同学更近一些，也相信你对此有不少的问题要问、不少的话要说。那我们就在这节答疑课里进一步沟通吧，我们从第 11 讲开始。

11 讲的答疑

思考题

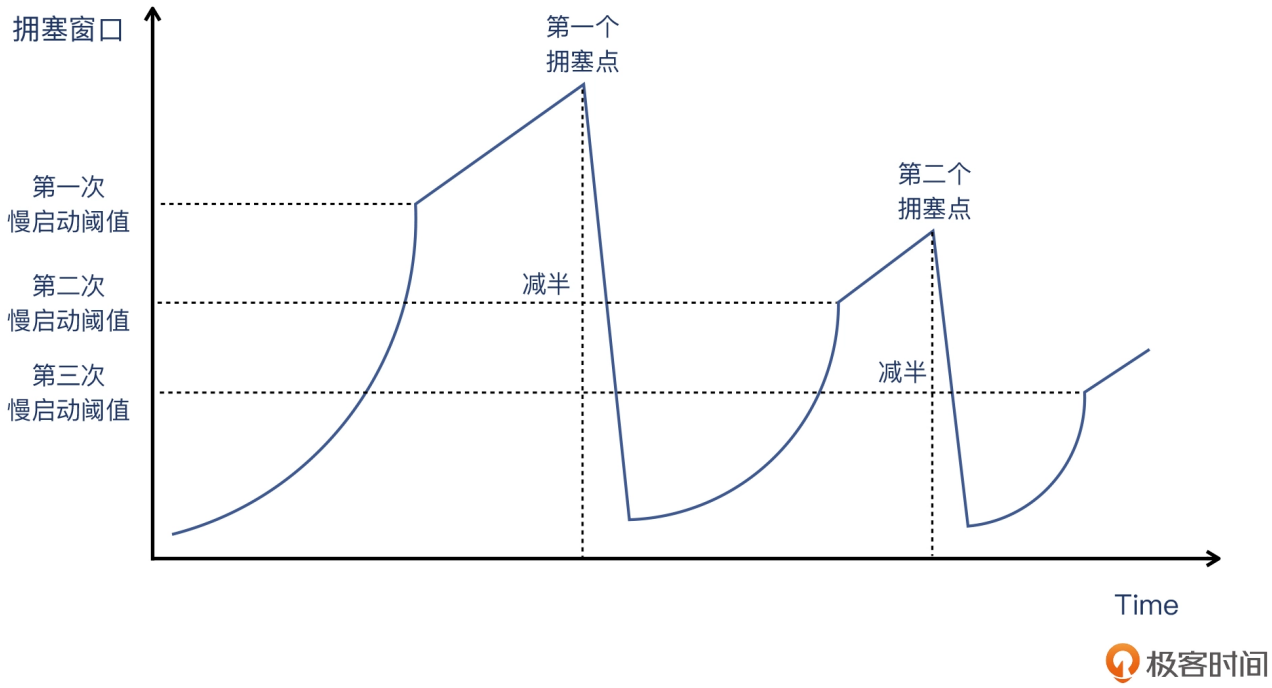
你在工作中有没有遇到拥塞引起的问题，或者有没有在抓包分析过程中，观察到过拥塞现象呢？

答案

这是一个开放式的问题，可以激发我们对这个话题的思考。可能是因为这一讲的内容理论性多过了实践性，所以留言不算多，但是每个问题都很有价值。比如 **@Chao** 同学的问题：

> 慢启动阈值（ssthresh）的初始值是如何确定的，当慢启动过程中没有进入重传状态，如何进入拥塞避免状态？

这确实是一个很好的问题。我在这一讲里（也包括其他很多资料）介绍的 TCP 慢启动过程，是下图这样的：



其中，第二次慢启动阈值是基于第一次拥塞窗口减半得到的。那么第一次慢启动阈值又是如何得到的呢？是不是也跟初始拥塞窗口一样，初始慢启动阈值也有一个默认值呢？

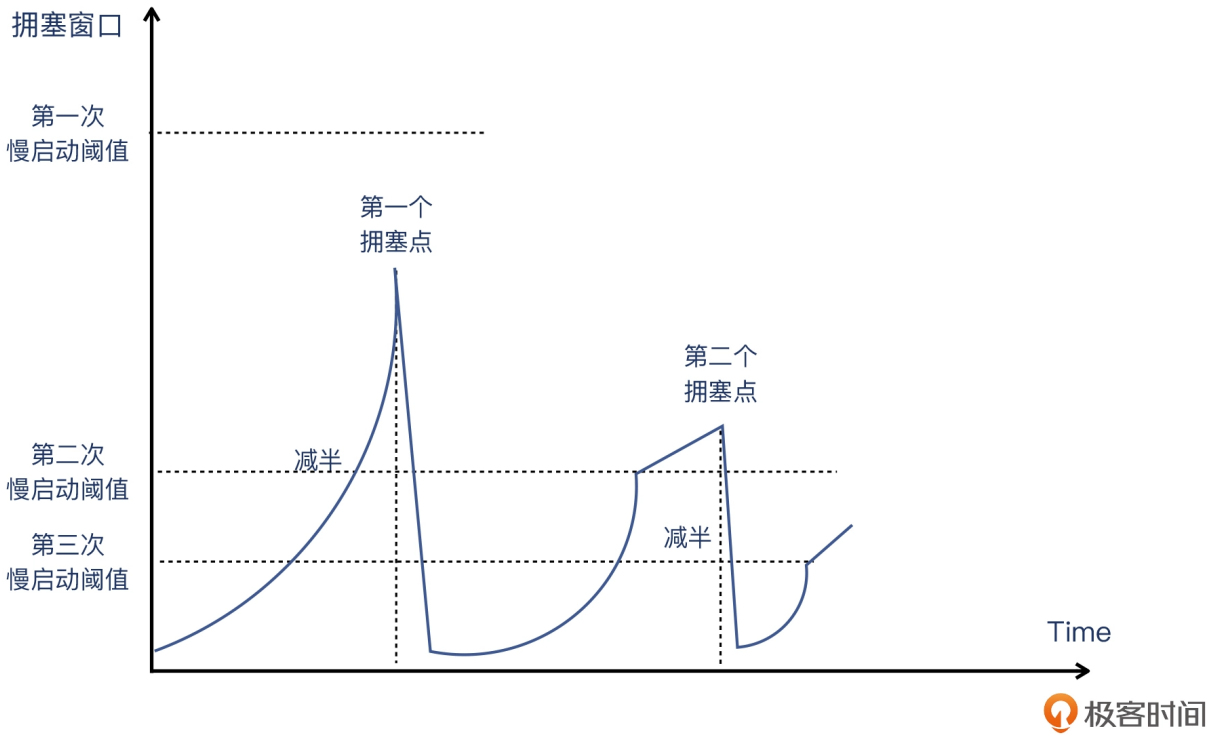
在具体的实现上，Linux（应该也包括其他 OS）的第一次慢启动阈值，实际上是“**无穷大**”。它被定义为：

```
1 #define TCP_INFINITE_SSTHRESH   0x7fffffff
```

复制代码

这个十六进制的 7fffffff 的十进制值，是 2147483647，也就是 2G。我们知道，发送窗口是拥塞窗口和对方的接收窗口之间的较小值。由于接收窗口理论最大值也只有 1G，因而发送窗口的最大值也是 1G，那么拥塞窗口超过 1G 也已经没有意义了，所以这里的 2G，事实上就是无穷大。

所以说，**在 Linux 的实现里，初始慢启动阈值肯定在第一个拥塞点之上。**这就造成一个现象：Linux 的 TCP 连接在慢启动后，先碰到的是拥塞点，而不是初始慢启动阈值。从现象上看就是，TCP 的拥塞窗口在慢启动过程中不断爬升，直到遇到第一个拥塞点（发生丢包或者超时），此时这个拥塞窗口的一半，就是第二次慢启动阈值了。示意图如下：



那么，Linux 为什么会选择这样的初始慢启动阈值呢？

主要原因，是当今的网络条件比多年前好了很多，所以为了“压榨”网络性能，让传输的启动阶段**尽可能快地达到理想的传输速度**，Linux 在传输刚开始还没有网络质量信息的时候，直接用了“碰到拥塞再快速恢复”的方式。这比预设一个折中的初始慢启动阈值的情况，会更快地达到理想的传输速度。

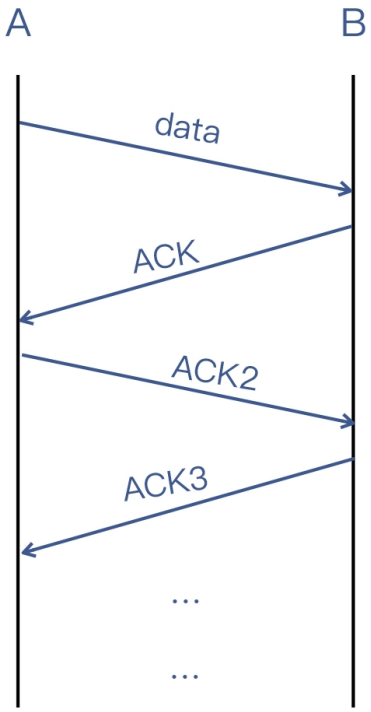
12 讲的答疑

思考题

1. TCP 的确认报文如果丢失了，发送端还会不会重传呢？为什么？
2. 你有没有遇到过重传引发的问题，你是怎么处理的呢？

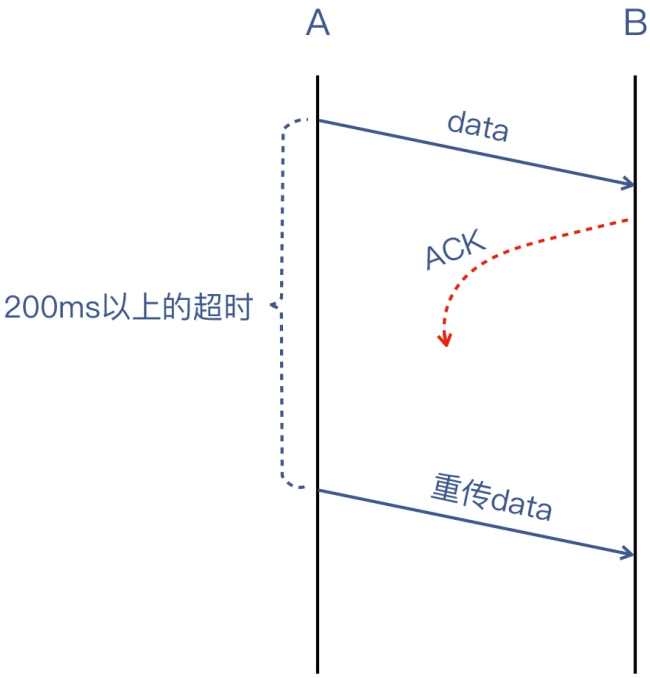
答案

在回答第一个问题之前，我们先思考另外一个问题：**TCP 对数据报文有确认机制，对确认报文也有确认机制吗？**比如是否会这样，就是 A 给 B 发一个数据报文，B 回复一个确认报文，A 再回复一个对确认报文的确认报文，然后 B 也如此操作一番……

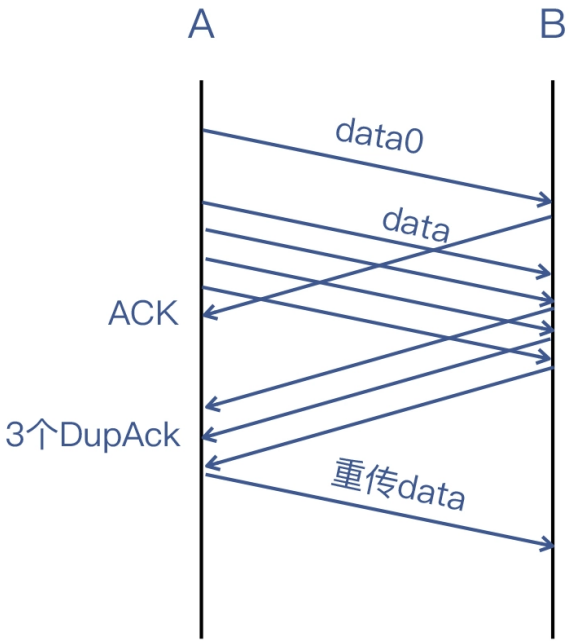


这就无穷无尽了，显然不是正确的做法。所以在 TCP 里，确认报文只发送一次。那你可能会问：“假如确认报文丢失了，那对端如何知道这件事呢？”

其实，这就是**超时重传**机制的用武之地了。比如，同样还是上面的图中，第一个 ACK 丢失的话，B 也不会重传这个 ACK，而 A 就会等待这个 ACK。一旦这个时间达到重传超时阈值（200ms 以上），A 就会启动超时重传机制。



当然，另一种很常见的情况是 A 还有更多的数据报文要发，于是情况就从超时重传，演变为快速重传：



第二个问题，**@ERROR404** 同学的回复是这样的：

> 重传还是看影不影响到业务，实际是允许重传存在的，比如像是互联网线路丢包，或者是营销等而引起流量突升的重传。

说得没错。其实重传也是 TCP 的必要特性，也正是**通过重传，TCP 实现了传输可靠性这一根本性的优势和特点**。反观 UDP，它就没有重传机制，当然也不能保证传输可靠性了。所以在 UDP 应用里，一般业务本身就是允许丢包的，或者在应用层自己实现传输可靠性，比如实现类似 TCP 重传的机制。

重传本身不是 TCP 的问题，可以说反而是它的特性，而引起重传的主要原因——丢包，才是真正的问题。这一般是链路质量导致的，常常要从 TCP 之外寻求答案。

关于重传的知识点，我在第 11~13 讲里都做了比较全面的介绍，希望对你建立起对重传的正确认识有所帮助。

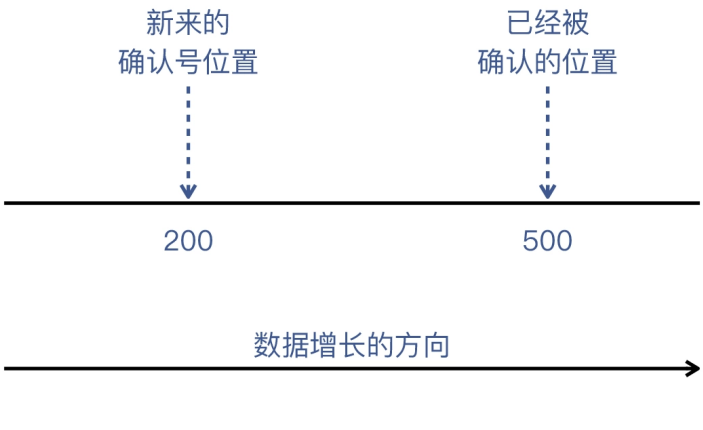
13 讲的答疑

思考题

1. 如果接收端收到一个确认包，其确认号为 200，而当前的被确认的位置在 500，那么接收端会怎么处理这个看起来“迟到并且重复”的确认包呢？
2. 你有没有遇到过这种“确认号在中间位置”的情况？当时有没有引起什么问题呢？

答案

第一个问题，就是说收到的报文确认了一个之前已经确认过的位置，如下图：



极客时间

这种情况，接收端可以“无视”。因为确认号是表示收到的连续字节的最新的位置，那么显然，一个数字接近但是更低的确认号（不是序列号回绕复用的那种情况），等于是一次信息的重复，接收端不会做任何处理。

> 补充：当然，因为序列号是 4G 之内循环使用的，当序列号越过 4G 后，下一个序列号就从低处开始了。不过因为这样的两个序列号离得远，并不是上面的那种情况。

第二个问题，确认号在中间的情况，暂时没有同学在留言区说遇到过。这样也挺好的，我们已经知道原理了，不来最好，来也不怕，对不对？

14 讲的答疑

思考题

1. “肉机”发出 100Mbps 的攻击流量，到达被攻击站点的时候，仍然是 100Mbps 吗？为什么呢？
2. 为什么 CDN 可以达到缓解 DDoS 的效果呢？

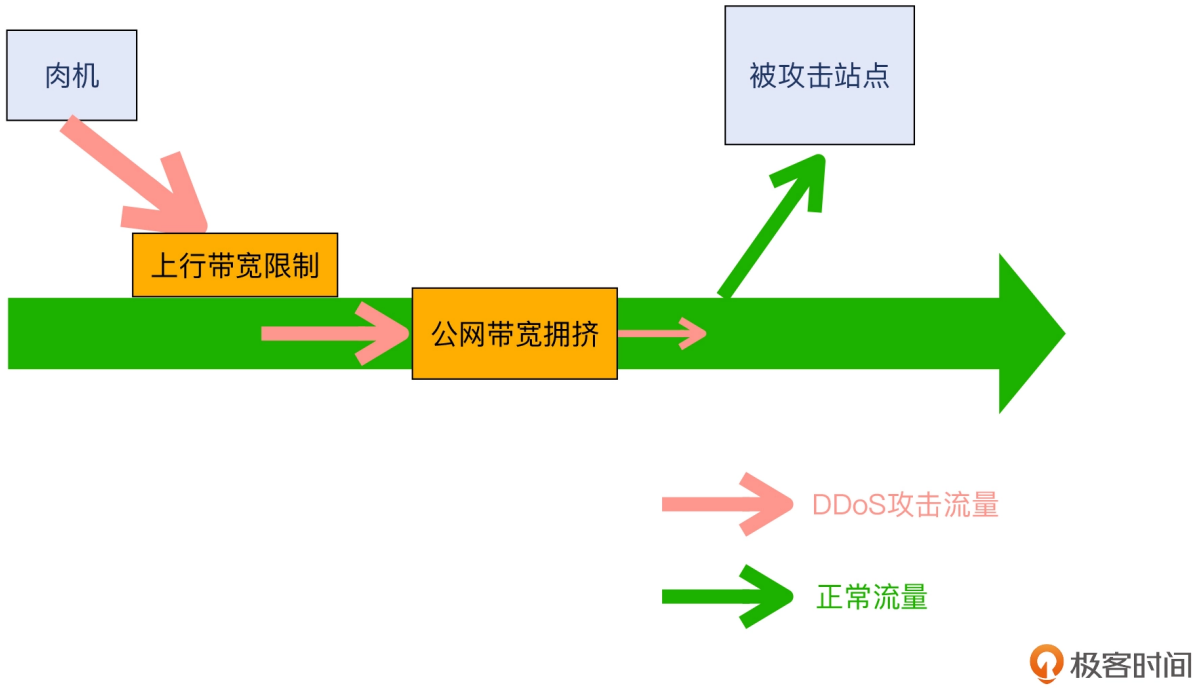
答案

第一个问题的核心，其实是关于公网流量的“衰减”问题。从网络路径来看，从“肉机”出来一直到达被攻击站点，途径的网络跳数一般也有 10 到 20 多跳。每一跳其实都有可能会遇到不同程度的衰减，比如以下这些：

“肉机”本身上行线路的**带宽限制**。

攻击流量在公网路径上因拥挤而导致的**流量损失**。

一般来说，“肉机”初始的攻击流量，在到达被攻击站点的时候，只有出发时的几分之一。因而作为攻击者，肯定要越发加大起始流量，这样经过公网衰减之后，还能有不少的流量到达被攻击站点。



第二个问题跟 **IP 数量和 DNS 解析**有关系。一般站点的 IP 就几个，而 CDN 因为是分布式的，一般能达到几十个 IP（几十个点）之多，而且不同地点解析到的 IP 也不同。这样的话，在 DDoS 攻击时，攻击流量也被分散到几十个 IP，这就被平均分散掉了。除了受影响严重的地区，其他地区依然能提供服务，这就在事实上起到了一定的防护效果。

当然，这并不是鼓励你要用 CDN 来扛 DDoS，而且它本身还有很多问题，比如：

这些 CDN 节点覆盖的地区失去了服务能力；

对这些 CDN 节点的其他客户产生了严重的影响。

那么，如果有人就是想把 CDN 当“廉价高防”来用，该怎么办呢？其实这个时候，CDN 服务商也不会默默地忍受，对于恶意使用 CDN 的情况，也是有可能“劝退客户”的。

15 讲的答疑

思考题

1. 第 7 个报文是 DupAck，为什么没有触发快速重传呢？
2. 消息网关那头的应用日志应该不是 499，那会是什么样的日志呢？

答案

第一个问题，这第 7 个报文确实是 DupAck，但是 DupAck 只有它一个，而快速重传的条件是什么？是**累积有 3 个或者以上的 DupAck**。

所以显然，单凭这一次的 DupAck 是无法触发快速重传的。

第二个问题，其实在课程里我有提到，就是：

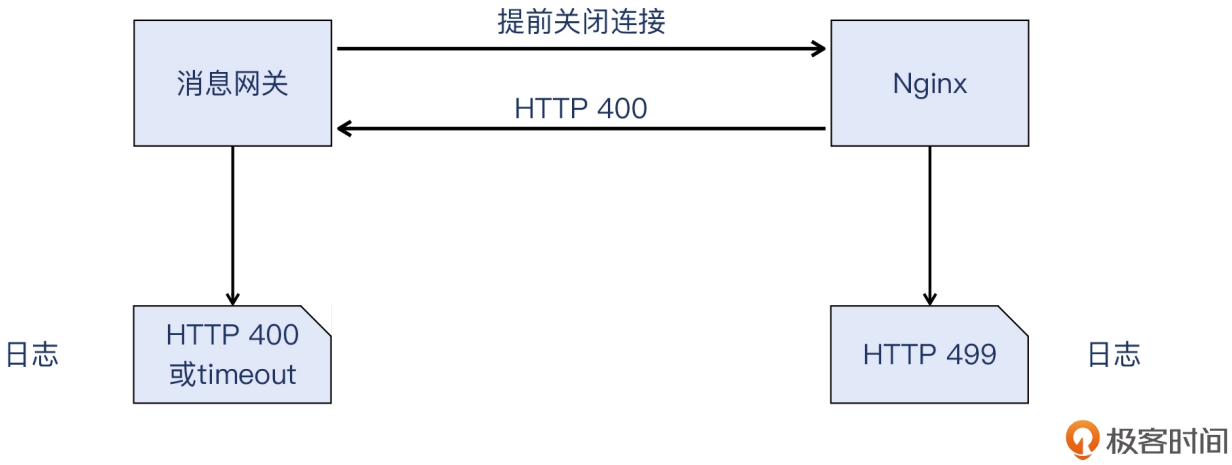
> 最后是报文 10，服务端发送了 HTTP 400 的响应报文给消息网关。

虽然作为服务端的 Nginx 记录在日志里的是 499，但是回复的 HTTP 响应，依然是遵循了标准 HTTP 协议，也就是回复了 HTTP 400。那么消息网关作为客户端，它一定是收到了 HTTP 400。

至于它是如何计入 Web 日志的，就是一个有意思的话题了。比如 Nginx 就没有记为 400 而是 499，那消息网关在日志里记录的，也未必一定是 400，也可能是它自己定义的某些值。所以说，这个问题的答案可能有两个：

记录的日志就是原始的 HTTP 400。

记录的是某种 timeout，如果它更关心的是业务状态，而不是 HTTP 响应。



好了，这次的答疑就到这里。你看完答案后有没有一些新的感悟呢？或者会不会有新的疑问呢？无论是哪种，都欢迎你在留言区跟大家一起分享，我们一同成长。

上一篇 答疑（二）| 第6~10讲思考题答案

下一篇 答疑（四）| 第16~20讲思考题答案

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。