

第11讲 | TCP协议（上）：因性恶而复杂，先恶后善反轻松

2018-06-11 刘超

趣谈网络协议

[进入课程 >](#)



讲述：刘超

时长 16:16 大小 7.49M



上一节，我们讲的 UDP，基本上包括了传输层所必须的端口字段。它就像我们小时候一样简单，相信“网之初，性本善，不丢包，不乱序”。

后来呢，我们都慢慢长大，了解了社会的残酷，变得复杂而成熟，就像 TCP 协议一样。它之所以这么复杂，那是因为它秉承的是“性恶论”。它天然认为网络环境是恶劣的，丢包、乱序、重传，拥塞都是常有的事情，一言不合就可能送达不了，因而要从算法层面来保证可靠性。

TCP 包头格式

我们先来看 TCP 头的格式。从这个图上可以看出，它比 UDP 复杂得多。

源端口号 (16位)					目的端口号 (16位)				
序号 (32位)									
确认序号 (32位)									
首部长度 (4位)	保留 (6位)	U R G	A C K	P S H	R S T	S Y N	F I N	窗口大小 (16位)	
校验和 (16位)					紧急指针 (16位)				
选项									
数据									

首先，源端口号和目标端口号是不可少的，这一点和 UDP 是一样的。如果没有这两个端口号。数据就不知道应该发给哪个应用。

接下来是包的序号。为什么要给包编号呢？当然是为了解决乱序的问题。不编好号怎么确认哪个应该先来，哪个应该后到呢。编号是为了解决乱序问题。既然是社会老司机，做事当然要稳重，一件件来，面临再复杂的情况，也临危不乱。

还应该有的就是确认序号。发出去的包应该有确认，要不然我怎么知道对方有没有收到呢？如果没有收到就应该重新发送，直到送达。这个可以解决不丢包的问题。作为老司机，做事当然要靠谱，答应了就要做到，暂时做不到也要有个回复。

TCP 是靠谱的协议，但是这不能说明它面临的网络环境好。从 IP 层面来讲，如果网络状况的确那么差，是没有任何可靠性保证的，而作为 IP 的上一层 TCP 也无能为力，唯一能做的就是更加努力，不断重传，通过各种算法保证。也就是说，对于 TCP 来讲，IP 层你丢不丢包，我管不着，但是我在我的层面上，会努力保证可靠性。

这有点像如果你在北京，和客户约十点见面，那么你应该清楚堵车是常态，你干预不了，也控制不了，你唯一能做的就是早走。打车不行就改乘地铁，尽力不失约。

接下来有一些状态位。例如 SYN 是发起一个连接，ACK 是回复，RST 是重新连接，FIN 是结束连接等。TCP 是面向连接的，因而双方要维护连接的状态，这些带状态位的包的发送，会引起双方的状态变更。

不像小时候，随便一个不认识的小朋友都能玩在一起，人大了，就变得礼貌，优雅而警觉，人与人遇到会互相热情的寒暄，离开会不舍的道别，但是人与人之间的信任会经过多次交互才能建立。

还有一个重要的就是窗口大小。TCP 要做流量控制，通信双方各声明一个窗口，标识自己当前能够的处理能力，别发送的太快，撑死我，也别发的太慢，饿死我。

作为老司机，做事情要有分寸，待人要把握尺度，既能适当提出自己的要求，又不强人所难。除了做流量控制以外，TCP 还会做拥塞控制，对于真正的通路堵车不堵车，它无能为力，唯一能做的就是控制自己，也即控制发送的速度。不能改变世界，就改变自己嘛。

作为老司机，要会自我控制，知进退，知道什么时候应该坚持，什么时候应该让步。

通过对 TCP 头的解析，我们知道要掌握 TCP 协议，重点应该关注以下几个问题：

顺序问题，稳重不乱；

丢包问题，承诺靠谱；

连接维护，有始有终；

流量控制，把握分寸；

拥塞控制，知进知退。

TCP 的三次握手

所有的问题，首先都要先建立一个连接，所以我们先来看连接维护问题。

TCP 的连接建立，我们常常称为三次握手。

A：您好，我是 A。

B：您好 A，我是 B。

A：您好 B。

我们也常称为“请求 -> 应答 -> 应答之应答”的三个回合。这个看起来简单，其实里面还是有很多的学问，很多的细节。

首先，为什么要三次，而不是两次？按说两个人打招呼，一来一回就可以了啊？为了可靠，为什么不是四次？

我们还是假设这个通路是非常不可靠的，A 要发起一个连接，当发了第一个请求杳无音信的时候，会有很多的可能性，比如第一个请求包丢了，再如没有丢，但是绕了弯路，超时了，还有 B 没有响应，不想和我连接。

A 不能确认结果，于是再发，再发。终于，有一个请求包到了 B，但是请求包到了 B 的这个事情，目前 A 还是不知道的，A 还有可能再发。

B 收到了请求包，就知道了 A 的存在，并且知道 A 要和它建立连接。如果 B 不乐意建立连接，则 A 会重试一阵后放弃，连接建立失败，没有问题；如果 B 是乐意建立连接的，则会发送应答包给 A。

当然对于 B 来说，这个应答包也是一入网络深似海，不知道能不能到达 A。这个时候 B 自然不能认为连接是建立好了，因为应答包仍然会丢，会绕弯路，或者 A 已经挂了都有可能。

而且这个时候 B 还能碰到一个诡异的现象就是，A 和 B 原来建立了连接，做了简单通信后，结束了连接。还记得吗？A 建立连接的时候，请求包重复发了几次，有的请求包绕了一大圈又回来了，B 会认为这也是一个正常的请求的话，因此建立了连接，可以想象，这个连接不会进行下去，也没有个终结的时候，纯属单相思了。因而两次握手肯定不行。

B 发送的应答可能会发送多次，但是只要一次到达 A，A 就认为连接已经建立了，因为对于 A 来讲，他的消息有去有回。A 会给 B 发送应答之应答，而 B 也在等这个消息，才能确认连接的建立，只有等到了这个消息，对于 B 来讲，才算它的消息有去有回。

当然 A 发给 B 的应答之应答也会丢，也会绕路，甚至 B 挂了。按理来说，还应该有个应答之应答之应答，这样下去就没底了。所以四次握手是可以的，四十次都可以，关键四百次也不能保证就真的可靠了。只要双方的消息都有去有回，就基本可以了。

好在大部分情况下，A 和 B 建立了连接之后，A 会马上发送数据的，一旦 A 发送数据，则很多问题都得到了解决。例如 A 发给 B 的应答丢了，当 A 后续发送的数据到达的时候，B 可以认为这个连接已经建立，或者 B 压根就挂了，A 发送的数据，会报错，说 B 不可达，A 就知道 B 出事情了。

当然你可以说 A 比较坏，就是不发数据，建立连接后空着。我们在程序设计的时候，可以要求开启 keepalive 机制，即使没有真实的数据包，也有探活包。

另外，你作为服务端 B 的程序设计者，对于 A 这种长时间不发包的客户端，可以主动关闭，从而空出资源来给其他客户端使用。

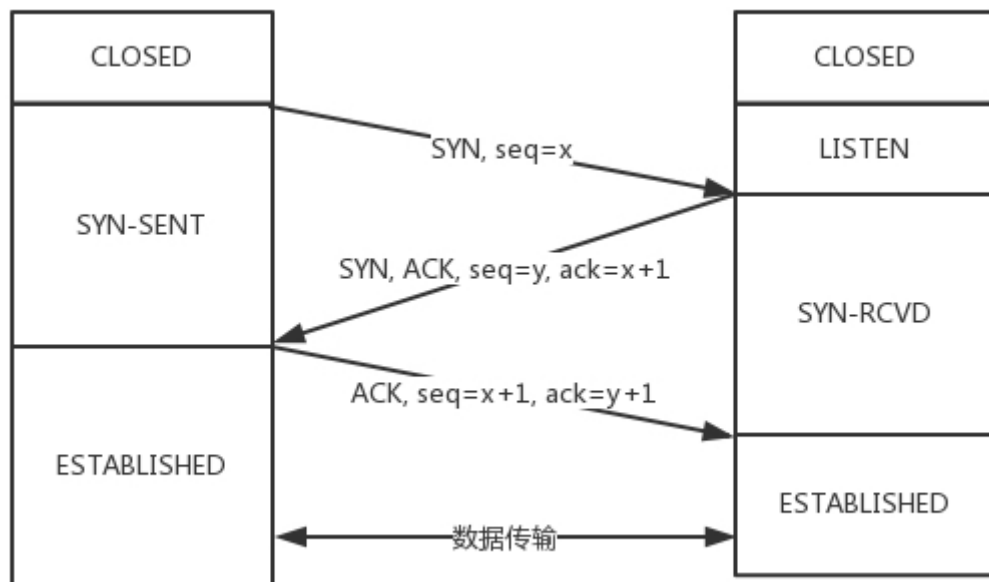
三次握手除了双方建立连接外，主要还是为了沟通一件事情，就是**TCP 包的序号的问题**。

A 要告诉 B，我这面发起的包的序号起始是从哪个号开始的，B 同样也要告诉 A，B 发起的包的序号起始是从哪个号开始的。为什么序号不能都从 1 开始呢？因为这样往往会出现冲突。

例如，A 连上 B 之后，发送了 1、2、3 三个包，但是发送 3 的时候，中间丢了，或者绕路了，于是重新发送，后来 A 掉线了，重新连上 B 后，序号又从 1 开始，然后发送 2，但是压根没想发送 3，但是上次绕路的那个 3 又回来了，发给了 B，B 自然认为，这就是下一个包，于是发生了错误。

因而，每个连接都要有不同的序号。这个序号的起始序号是随着时间变化的，可以看成一个 32 位的计数器，每 4ms 加一，如果计算一下，如果到重复，需要 4 个多小时，那个绕路的包早就死翘翘了，因为我们都知道 IP 包头里面有个 TTL，也即生存时间。

好了，双方终于建立了信任，建立了连接。前面也说过，为了维护这个连接，双方都要维护一个状态机，在连接建立的过程中，双方的状态变化时序图就像这样。



一开始，客户端和服务端都处于 CLOSED 状态。先是服务端主动监听某个端口，处于 LISTEN 状态。然后客户端主动发起连接 SYN，之后处于 SYN-SENT 状态。服务端收到发起的连接，返回 SYN，并且 ACK 客户端的 SYN，之后处于 SYN-RCVD 状态。客户端收到服务端发送的 SYN 和 ACK 之后，发送 ACK 的 ACK，之后处于 ESTABLISHED 状态，因为它一发一收成功了。服务端收到 ACK 的 ACK 之后，处于 ESTABLISHED 状态，因为它也一发一收了。

TCP 四次挥手

好了，说完了连接，接下来说一说“拜拜”，好说好散。这常被称为四次挥手。

A：B 啊，我不想玩了。

B：哦，你不想玩了啊，我知道了。

这个时候，还只是 A 不想玩了，也即 A 不会再发送数据，但是 B 能不能在 ACK 的时候，直接关闭呢？当然不可以了，很有可能 A 是发完了最后的数据就准备不玩了，但是 B 还没做完自己的事情，还是可以发送数据的，所以称为半关闭的状态。

这个时候 A 可以选择不再接收数据了，也可以选择最后再接收一段数据，等待 B 也主动关闭。

B：A 啊，好吧，我也不玩了，拜拜。

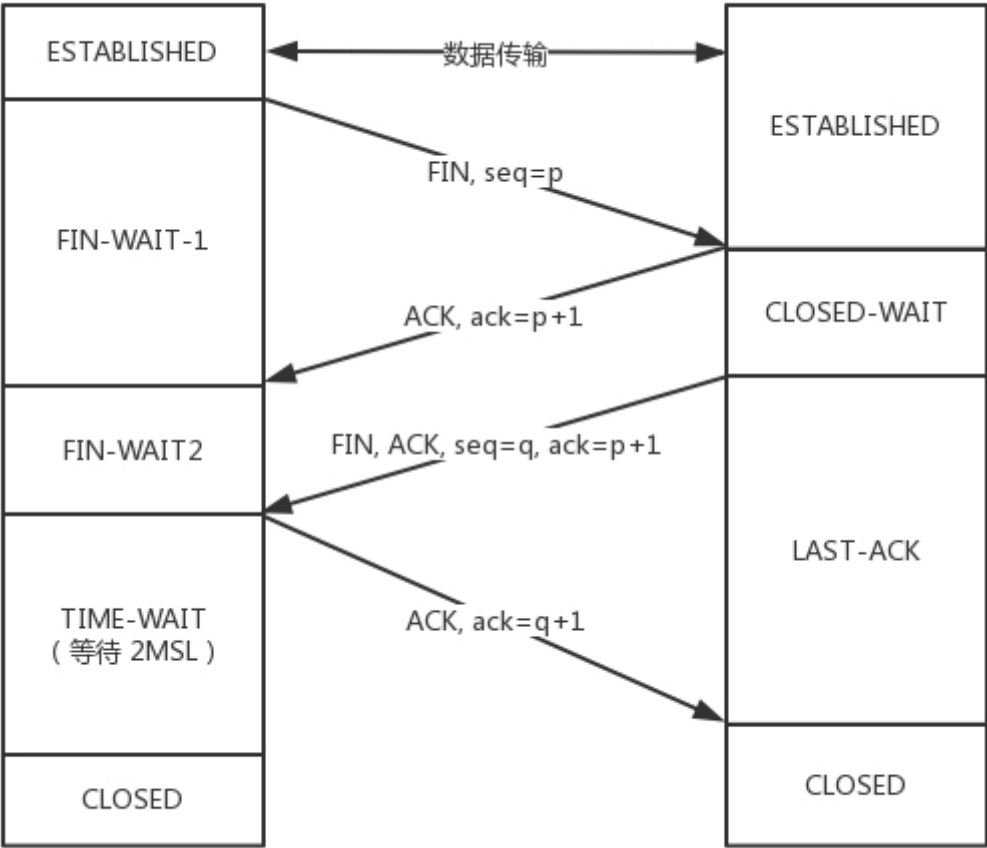
A：好的，拜拜。

这样整个连接就关闭了。但是这个过程有没有异常情况呢？当然有，上面是和平分手的场面。

A 开始说“不玩了”，B 说“知道了”，这个回合，是没什么问题的，因为在此之前，双方还处于合作的状态，如果 A 说“不玩了”，没有收到回复，则 A 会重新发送“不玩了”。但是这个回合结束之后，就有可能出现异常情况了，因为已经有一方率先撕破脸。

一种情况是，A 说完“不玩了”之后，直接跑路，是会有问题的，因为 B 还没有发起结束，而如果 A 跑路，B 就算发起结束，也得不到回答，B 就不知道该怎么办了。另一种情况是，A 说完“不玩了”，B 直接跑路，也是有问题的，因为 A 不知道 B 是还有事情要处理，还是过一会儿会发送结束。

那怎么解决这些问题呢？TCP 协议专门设计了几个状态来处理这些问题。我们来看断开连接的时候的**状态时序图**。



断开的时候，我们可以看到，当 A 说“不玩了”，就进入 FIN_WAIT_1 的状态，B 收到“A 不玩”的消息后，发送知道了，就进入 CLOSE_WAIT 的状态。

A 收到“B 说知道了”，就进入 FIN_WAIT_2 的状态，如果这个时候 B 直接跑路，则 A 将永远在这个状态。TCP 协议里面并没有对这个状态的处理，但是 Linux 有，可以调整 tcp_fin_timeout 这个参数，设置一个超时时间。

如果 B 没有跑路，发送了“B 也不玩了”的请求到达 A 时，A 发送“知道 B 也不玩了”的 ACK 后，从 FIN_WAIT_2 状态结束，按说 A 可以跑路了，但是最后的这个 ACK 万一 B 收不到呢？则 B 会重新发一个“B 不玩了”，这个时候 A 已经跑路了的话，B 就再也收不到 ACK 了，因而 TCP 协议要求 A 最后等待一段时间 TIME_WAIT，这个时间要足够长，长到如果 B 没收到 ACK 的话，“B 说不玩了”会重发的，A 会重新发一个 ACK 并且足够时间到达 B。

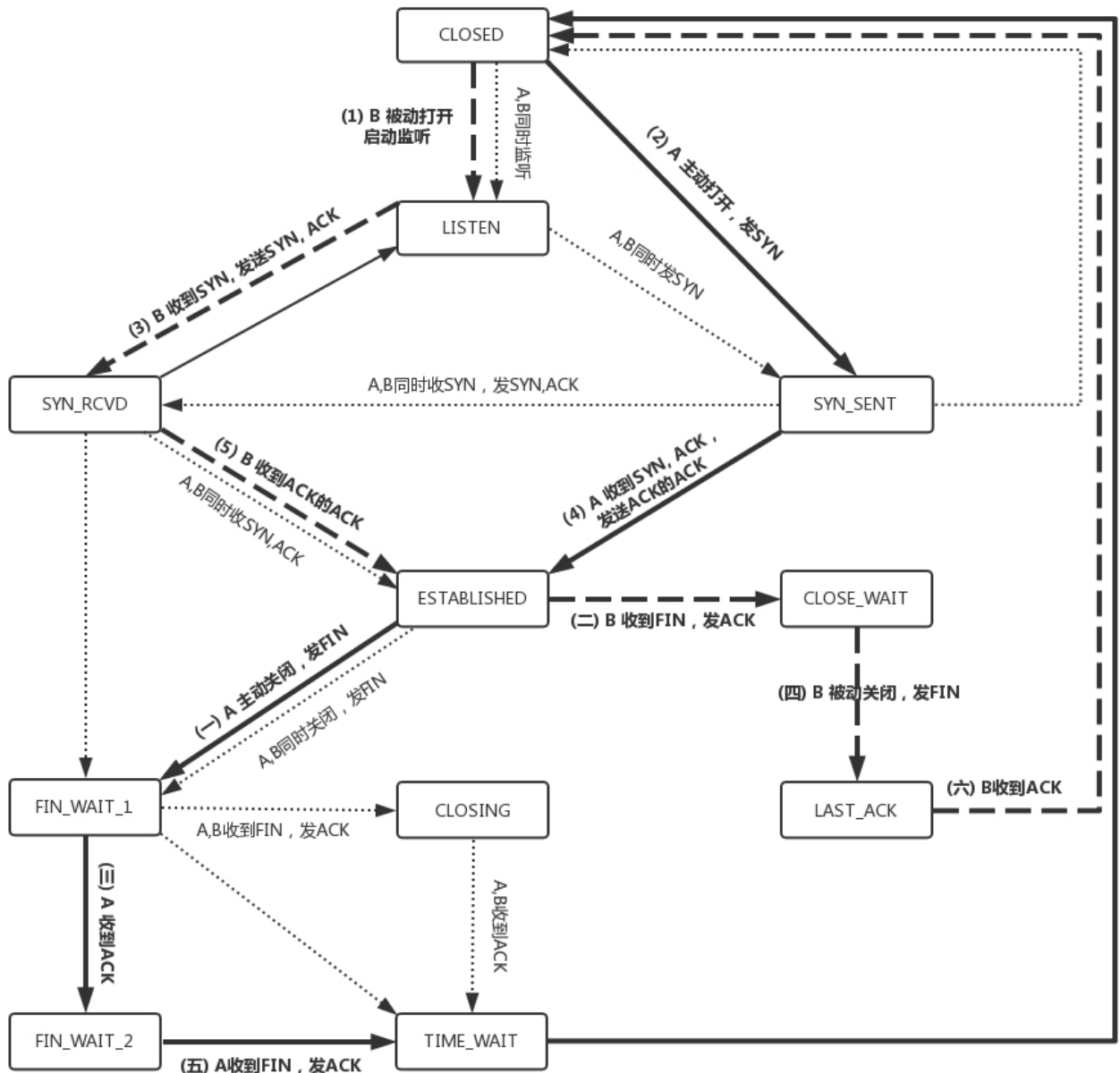
A 直接跑路还有一个问题是，A 的端口就直接空出来了，但是 B 不知道，B 原来发过的很多包很可能还在路上，如果 A 的端口被一个新的应用占用了，这个新的应用会收到上个连接中 B 发过来的包，虽然序列号是重新生成的，但是这里要上一个双保险，防止产生混乱，因而也需要等足够长的时间，等到原来 B 发送的所有的包都死翘翘，再空出端口来。

等待的时间设为 2MSL，MSL 是 **Maximum Segment Lifetime**，**报文最大生存时间**，它是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃。因为 TCP 报文基于 IP 协议的，而 IP 头中有一个 TTL 域，是 IP 数据报可以经过的最大路由数，每经过一个处理他的路由器此值就减 1，当此值为 0 则数据报将被丢弃，同时发送 ICMP 报文通知源主机。协议规定 MSL 为 2 分钟，实际应用中常用的是 30 秒，1 分钟和 2 分钟等。

还有一个异常情况就是，B 超过了 2MSL 的时间，依然没有收到它发的 FIN 的 ACK，怎么办呢？按照 TCP 的原理，B 当然还会重发 FIN，这个时候 A 再收到这个包之后，A 就表示，我已经在这里等了这么长时间了，已经仁至义尽了，之后的我就都不认了，于是就直接发送 RST，B 就知道 A 早就跑了。

TCP 状态机

将连接建立和连接断开的两个时序状态图综合起来，就是这个著名的 TCP 的状态机。学习的时候比较建议将这个状态机和时序状态机对照着看，不然容易晕。



在这个图中，加黑加粗的部分，是上面说到的主要流程，其中阿拉伯数字的序号，是连接过程中的顺序，而大写中文数字的序号，是连接断开过程中的顺序。加粗的实线是客户端 A 的状态变迁，加粗的虚线是服务端 B 的状态变迁。

小结

好了，这一节就到这里了，我来做一个总结：

TCP 包头很复杂，但是主要关注五个问题，顺序问题，丢包问题，连接维护，流量控制，拥塞控制；

连接的建立是经过三次握手，断开的时候四次挥手，一定要掌握的我画的那个状态图。

最后，给你留两个思考题。

1. TCP 的连接有这么多的状态，你知道如何在系统中查看某个连接的状态吗？
2. 这一节仅仅讲了连接维护问题，其实为了维护连接的状态，还有其他的数据结构来处理其他的四个问题，那你知道是什么吗？

欢迎你留言和我讨论。趣谈网络协议，我们下期见！

 极客时间

趣谈网络协议

像小说一样的网络协议入门课

刘超 网易研究院
云计算技术部首席架构师



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 第10讲 | UDP协议：因性善而简单，难免碰到“城会玩”

下一篇 第12讲 | TCP协议（下）：西行必定多妖孽，恒心智慧消磨难

精选留言 (100)

写留言



code4j 置顶

2018-06-28

13

老师您好，昨天阿里云出故障，恢复后，我们调用阿里云服务的时后出现了调用出异常 connection reset。netstat看了下这个ip发现都是timewait，链接不多，但是始终无法连

播放对方的服务。按照今天的内容，难道是我的程序关闭主动关闭链接后没有发出最后的ack吗？之前都没有问题，很不解

展开 ▾



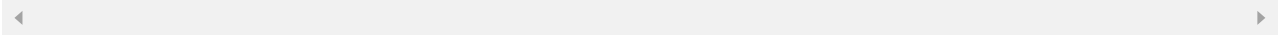
进阶的码农 置顶

2018-06-11

👍 4

状态机图里的不加粗虚线看不懂什么意思 麻烦老师点拨下

作者回复: 其他非主流过程



monkay

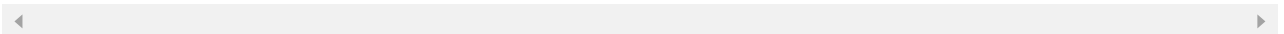
2018-06-11

👍 27

如果是建立链接了，数据传输过程链接断了，客户端和服务端各自会是什么状态？或者我可以这样理解么，所谓的链接根本是不存在的，双方握手之后，数据传输还是跟udp一样，只是tcp在维护顺序、流量之类的控制

展开 ▾

作者回复: 是的，连接就是两端的状态维护，中间过程没有所谓的连接，一旦传输失败，一端收到消息，才知道状态的变化



krugle

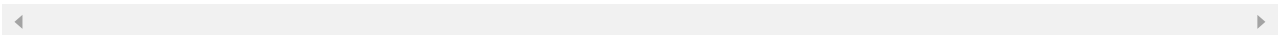
2018-08-17

👍 25

流量控制和拥塞控制什么区别

展开 ▾

作者回复: 一个是对另一端的，一个是针对网络的



Ender0224

2018-06-11

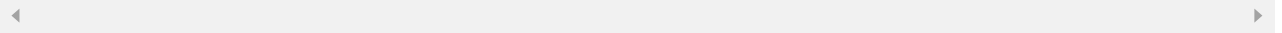
👍 16

多谢分享，精彩。扫除了我之前很多的疑问。tcp连接的断开比建立复杂一些，本质上是因为资源的申请（初始化）本身就比较资源的释放简单，以c++为例，构造函数初始化对象很

简单，而析构函数则要考虑所有资源安全有序的释放，tcp断连时序中除了断开这一重要动作，另外重要的潜台词是“我要断开连接了 你感觉把收尾工作做了”

展开 ∨

作者回复: 谢谢



Colin.Tao

2018-09-23

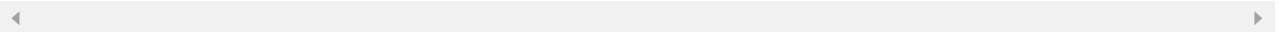
👍 14

旧书不厌百回读，熟读深思子自知啊。

再一次认识下TCP这位老司机，这可能是我读过的最好懂的讲TCP的文章了。同时有了一个很爽的触点，我发现但凡复杂点儿的东西，状态数据都复杂很多。还有一个，也是最重要的，我从TCP中再一次认识到了一个做人的道理，像孔子说的：“不怨天，不尤人”。人...

展开 ∨

作者回复: 赞



小田

2018-06-12

👍 12

人大了，就变得礼貌，优雅而警觉

展开 ∨



eason2017

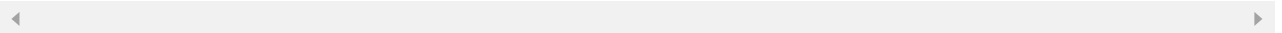
2018-07-04

👍 11

老师好，我看书中说计数器每4微妙就加1的。

展开 ∨

作者回复: 赞



零一

2018-06-11

👍 9

可以用 netstat 或者 lsof 命令 grep 一下 establish listen close_wait 等这些查看



Ender0224

2018-06-14

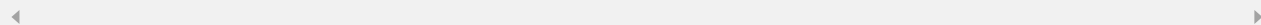
👍 8

评论区不能互评，如下两个问题是我的看法，不对请指出 多谢

我们做一个基于tcp的“物联网”应用（中国移动网络），如上面所说tcp层面已经会自动重传数据了，业务层面上还有必要再重传吗？如果是的话，业务需要多久重传一次？...

展开 ∨

作者回复: 赞



Adolph

2019-03-27

👍 7

为什么要三次握手

三次握手的目的是建立可靠的通信信道，说到通讯，简单来说就是数据的发送与接收，而三次握手最主要的目的就是双方确认自己与对方的发送与接收是正常的。

第一次握手：Client 什么都不能确认；Server 确认了对方发送正常...

展开 ∨



hquery

2018-07-24

👍 6

敲黑板，这是面试重点

展开 ∨



HunterYuan

2018-07-14

👍 6

老师回答，@正是那朵玫瑰的第一个问题，感觉稍微有点问题，我理解，不是每次数据交互的会发ack，有些ack是可以合并的，用于减少数据包量，通过wireshark抓包，也证实了。理由是，因为ack号是表示已收到的数据量，也就是说，它是告诉发送方目前已接收的数据的最后一个位置在哪里，因此当需要连续发送ack时，只要发送最后一个ack号就可以

了，中间的可以全部省略。同样的机制还有ack号通知和窗口更新通知合并。这是我的理...
展开 ▾



刘宝明

2018-06-11

👍 5

老师可以再教程里加一点小实验吗

展开 ▾



进化论

2018-06-11

👍 4

keepalive机制老师能深入讲解下吗

展开 ▾



姜戈

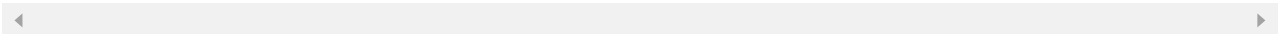
2018-06-11

👍 4

Tcp三次握手设计被恶意利用，造就了ddos。

展开 ▾

作者回复: 是有连接攻击的



babos

2018-07-02

👍 3

四次挥手是不是可以做成三次，把第二步和第三步的ACK和FIN一起发

因为b可能还有一些事情没做完，需要做完了才能结束

展开 ▾



正是那朵玫...

2018-06-12

👍 3

老师有几个疑问：

1、当三次握手建立连接后，每次数据交互都还会ack吗？比如建立连接后，客户端发送数据包给服务器端，服务器成功收到数据包后会发送ack给客户端么？

2、如果建立连接后，客户端和服务端没有任何数据交互，双方也不主动关闭连接，理论上这个连接会一直存在么？...

展开 ▾

作者回复: 是的，每次都ack。可以有keepalive，如果不交互，两边都不像释放，那就数据结构一直占用内存，对网络没啥影响。必须是发送数据的时候，才知道跑路的事情。你所谓的强制关是怎么关？有可能还是发送了fin的



洪宇舟

2018-06-11

👍 3

刘老师，最后的状态机中，细虚线表示什么呢？

展开 ▾



William

2018-11-23

👍 2

感谢老师耐心回答，另外RFC文档上，microsecond是微秒哈，毫秒是milisecond 😊

作者回复: 哦，谢谢指正

