

10 | 如何提升TCP四次挥手的性能?

2020-05-20 陶辉

系统性能调优必知必会

[进入课程 >](#)



讲述：陶辉

时长 17:23 大小 15.92M



你好，我是陶辉。

上一节课，我们介绍了建立连接时的优化方法，这一节课再来看四次挥手关闭连接时，如何优化性能。

close 和 shutdown 函数都可以关闭连接，但这两种方式关闭的连接，不只功能上有差异，控制它们的 Linux 参数也不相同。close 函数会让连接变为孤儿连接，shutdown 函数则允许在半关闭的连接上长时间传输数据。TCP 之所以具备这个功能，是因为它是全双工协议，但这也造成四次挥手非常复杂。



四次挥手中你可以用 `netstat` 命令观察到 6 种状态。其中，你多半看到过 `TIME_WAIT` 状态。网上有许多文章介绍怎样减少 `TIME_WAIT` 状态连接的数量，也有文章说 `TIME_WAIT` 状态是必不可少、不能优化掉的。这两种看似自相矛盾的观点之所以存在，就在于优化连接关闭时，不能仅基于主机端的视角，还必须站在整个网络的层次上，才能给出正确的解决方案。

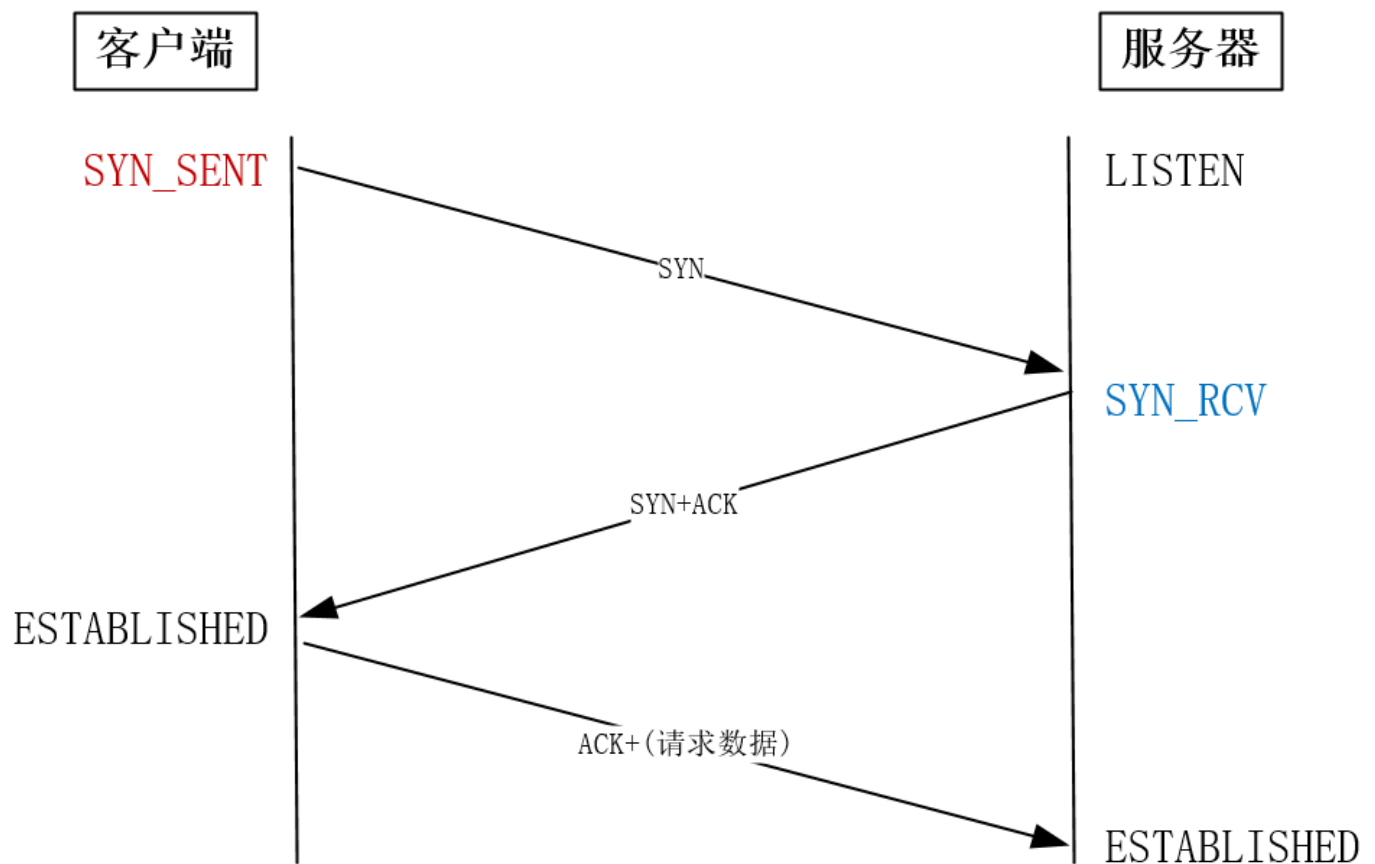
Linux 为四次挥手提供了很多控制参数，有些参数的名称与含义并不相符。例如 `tcp_orphan_retries` 参数中有 `orphan` 孤儿，却同时对非孤儿连接也生效。而且，错误地配置这些参数，不只无法针对高并发场景提升性能，还会降低资源的使用效率，甚至引发数据错误。

这一讲，我们将基于四次挥手的流程，介绍 Linux 下的优化方法。

四次挥手的流程

你想没想过，为什么建立连接是三次握手，而关闭连接需要四次挥手呢？

这是因为 TCP 不允许连接处于半打开状态时就单向传输数据，所以在三次握手建立连接时，服务器会把 `ACK` 和 `SYN` 放在一起发给客户端，其中，`ACK` 用来打开客户端的发送通道，`SYN` 用来打开服务器的发送通道。这样，原本的四次握手就降为三次握手了。

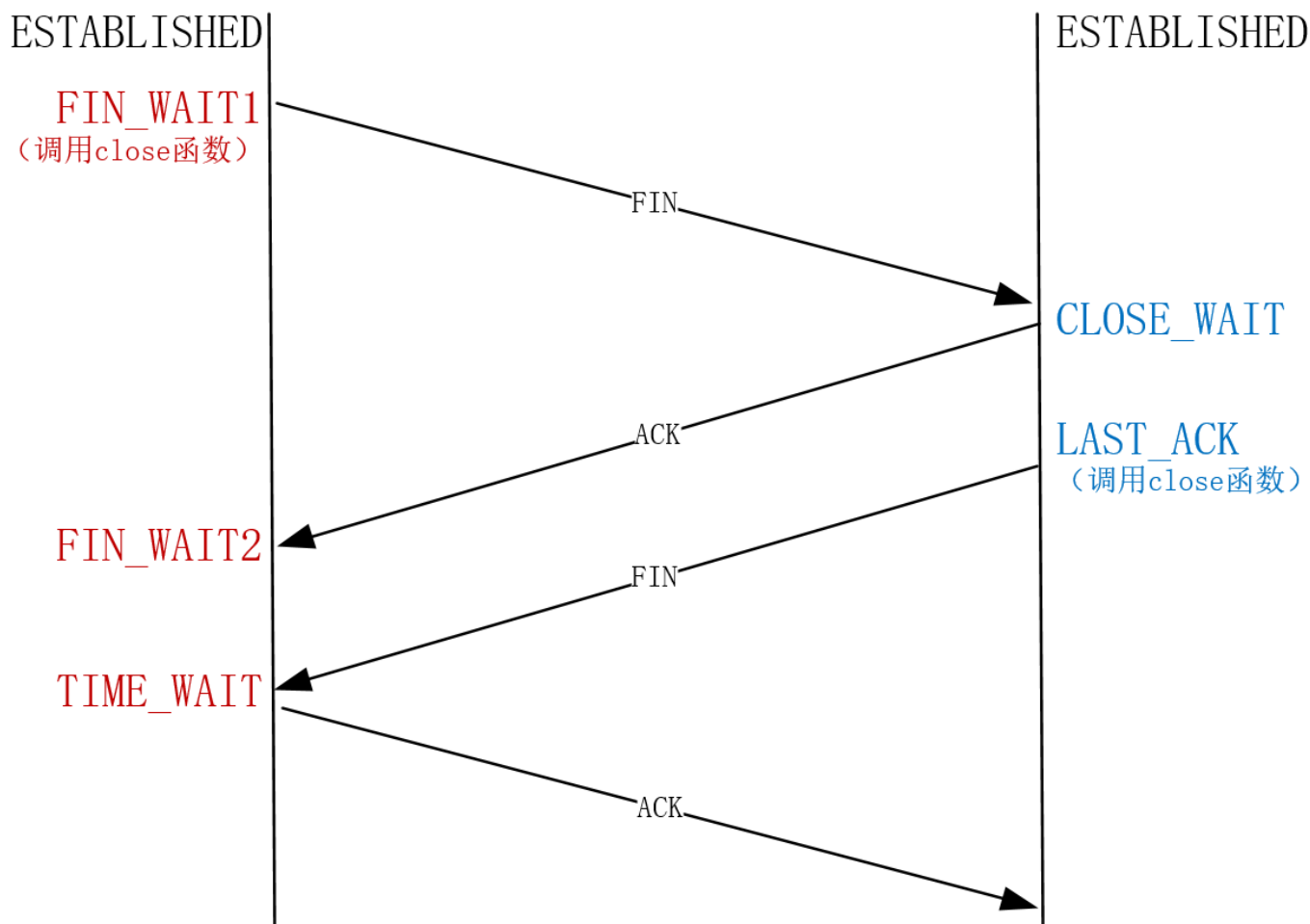


但是当连接处于半关闭状态时，TCP 是允许单向传输数据的。为便于下文描述，**接下来我们把先关闭连接的一方叫做主动方，后关闭连接的一方叫做被动方**。当主动方关闭连接时，被动方仍然可以在不调用 `close` 函数的状态下，长时间发送数据，此时连接处于半关闭状态。这一特性是 TCP 的双向通道互相独立所致，却也使得关闭连接必须通过四次挥手才能做到。

互联网中往往服务器才是主动关闭连接的一方。这是因为，HTTP 消息是单向传输协议，服务器接收完请求才能生成响应，发送完响应后就会立刻关闭 TCP 连接，这样及时释放了资源，能够为更多的用户服务。

这就使得服务器的优化策略变得复杂起来。一方面，由于被动方有多种应对策略，从而增加了主动方的处理分支。另一方面，服务器同时为成千上万个用户服务，任何错误都会被庞大的用户数放大。所以对主动方的关闭连接参数调整时，需要格外小心。

了解了这一点之后，我们再来看四次挥手的流程。



其实四次挥手只涉及两种报文：FIN 和 ACK。FIN 就是 Finish 结束连接的意思，谁发出 FIN 报文，就表示它将不再发送任何数据，关闭这一方向的传输通道。ACK 是 Acknowledge 确认的意思，它用来通知对方：你方的发送通道已经关闭。

当主动方关闭连接时，会发送 FIN 报文，此时主动方的连接状态由 ESTABLISHED 变为 FIN_WAIT1。当被动方收到 FIN 报文后，内核自动回复 ACK 报文，连接状态由 ESTABLISHED 变为 CLOSE_WAIT，顾名思义，它在等待进程调用 close 函数关闭连接。当主动方接收到这个 ACK 报文后，连接状态由 FIN_WAIT1 变为 FIN_WAIT2，主动方的发送通道就关闭了。

再来看被动方的发送通道是如何关闭的。当被动方进入 CLOSE_WAIT 状态时，进程的 read 函数会返回 0，这样开发人员就会有针对性地调用 close 函数，进而触发内核发送 FIN 报文，此时被动方连接的状态变为 LAST_ACK。当主动方收到这个 FIN 报文时，内核会自动回复 ACK，同时连接的状态由 FIN_WAIT2 变为 TIME_WAIT，Linux 系统下大约 1 分钟后 TIME_WAIT 状态的连接才会彻底关闭。而被动方收到 ACK 报文后，连接就会关闭。

主动方的优化

关闭连接有多种方式，比如进程异常退出时，针对它打开的连接，内核就会发送 RST 报文来关闭。RST 的全称是 Reset 复位的意思，它可以不走四次挥手强行关闭连接，但当报文延迟或者重复传输时，这种方式会导致数据错乱，所以这是不得已而为之的关闭连接方案。


安全关闭连接的方式必须通过四次挥手，它由进程调用 close 或者 shutdown 函数发起，这二者都会向对方发送 FIN 报文（shutdown 参数须传入 SHUT_WR 或者 SHUT_RDWR 才会发送 FIN），区别在于 close 调用后，哪怕对方在半关闭状态下发送的数据到达主动方，进程也无法接收。

此时，这个连接叫做孤儿连接，如果你用 netstat -p 命令，会发现连接对应的进程名为空。而 shutdown 函数调用后，即使连接进入了 FIN_WAIT1 或者 FIN_WAIT2 状态，它也不是孤儿连接，进程仍然可以继续接收数据。关于孤儿连接的概念，下文调优参数时还会用到。

主动方发送 FIN 报文后，连接就处于 FIN_WAIT1 状态下，该状态通常应在数十毫秒内转为 FIN_WAIT2。只有迟迟收不到对方返回的 ACK 时，才能用 netstat 命令观察到 FIN_WAIT1 状态。此时，**内核会定时重发 FIN 报文，其中重发次数由**

tcp_orphan_retries 参数控制（注意，orphan 虽然是孤儿的意思，该参数却不只对孤儿连接有效，事实上，它对所有 FIN_WAIT1 状态下的连接都有效），默认值是 0，特指 8 次：

```
1 net.ipv4.tcp_orphan_retries = 0
```

 复制代码

如果 FIN_WAIT1 状态连接有很多，你就需要考虑降低 tcp_orphan_retries 的值。当重试次数达到 tcp_orphan_retries 时，连接就会直接关闭掉。


对于正常情况来说，调低 tcp_orphan_retries 已经够用，但如果遇到恶意攻击，FIN 报文根本无法发送出去。这是由 TCP 的 2 个特性导致的。

首先，TCP 必须保证报文是有序发送的，FIN 报文也不例外，当发送缓冲区还有数据没发送时，FIN 报文也不能提前发送。

其次，TCP 有流控功能，当接收方将接收窗口设为 0 时，发送方就不能再发送数据。所以，当攻击者下载大文件时，就可以通过将接收窗口设为 0，导致 FIN 报文无法发送，进而导致连接一直处于 FIN_WAIT1 状态。

解决这种问题的方案是调整 tcp_max_orphans 参数：


```
1 net.ipv4.tcp_max_orphans = 16384
```

 复制代码

顾名思义，**tcp_max_orphans 定义了孤儿连接的最大数量**。当进程调用 close 函数关闭连接后，无论该连接是在 FIN_WAIT1 状态，还是确实关闭了，这个连接都与该进程无关了，它变成了孤儿连接。Linux 系统为防止孤儿连接过多，导致系统资源长期被占用，就提供了 tcp_max_orphans 参数。如果孤儿连接数量大于它，新增的孤儿连接将不再走四次挥手，而是直接发送 RST 复位报文强制关闭。

当连接收到 ACK 进入 FIN_WAIT2 状态后，就表示主动方的发送通道已经关闭，接下来将等待对方发送 FIN 报文，关闭对方的发送通道。这时，**如果连接是用 shutdown 函数关闭的，连接可以一直处于 FIN_WAIT2 状态。但对于 close 函数关闭的孤儿连接，这个状态不可以持续太久，而 tcp_fin_timeout 控制了这个状态下连接的持续时长。**

```
1 net.ipv4.tcp_fin_timeout = 60
```

 复制代码

它的默认值是 60 秒。这意味着对于孤儿连接，如果 60 秒后还没有收到 FIN 报文，连接就会直接关闭。这个 60 秒并不是拍脑袋决定的，它与接下来介绍的 TIME_WAIT 状态的持续时间是相同的，我们稍后再来回答 60 秒的由来。

TIME_WAIT 是主动方四次挥手的最后一个状态。当收到被动方发来的 FIN 报文时，主动方回复 ACK，表示确认对方的发送通道已经关闭，连接随之进入 TIME_WAIT 状态，等待 60 秒后关闭，为什么呢？我们必须站在整个网络的角度上，才能回答这个问题。

TIME_WAIT 状态的连接，在主动方看来确实已经关闭了。然而，被动方没有收到 ACK 报文前，连接还处于 LAST_ACK 状态。如果这个 ACK 报文没有到达被动方，被动方就会重

发 FIN 报文。重发次数仍然由前面介绍过的 `tcp_orphan_retries` 参数控制。

如果主动方不保留 `TIME_WAIT` 状态，会发生什么呢？此时连接的端口恢复了自由身，可以复用于新连接了。然而，被动方的 FIN 报文可能再次到达，这既可能是网络中的路由器重复发送，也有可能是被动方没收到 ACK 时基于 `tcp_orphan_retries` 参数重发。这样，**正常通讯的新连接就可能被重复发送的 FIN 报文误关闭**。保留 `TIME_WAIT` 状态，就可以应付重发的 FIN 报文，当然，其他数据报文也有可能重发，所以 `TIME_WAIT` 状态还能避免数据错乱。

我们再回过头来看看，为什么 `TIME_WAIT` 状态要保持 60 秒呢？这与孤儿连接 `FIN_WAIT2` 状态默认保留 60 秒的原理是一样的，**因为这两个状态都需要保持 2MSL 时长。MSL 全称是 Maximum Segment Lifetime，它定义了一个报文在网络中的最长生存时间**（报文每经过一次路由器的转发，IP 头部的 TTL 字段就会减 1，减到 0 时报文就被丢弃，这就限制了报文的最长存活时间）。

为什么是 2 MSL 的时长呢？这其实是相当于至少允许报文丢失一次。比如，若 ACK 在一个 MSL 内丢失，这样被动方重发的 FIN 会在第 2 个 MSL 内到达，`TIME_WAIT` 状态的连接可以应对。为什么不是 4 或者 8 MSL 的时长呢？你可以想象一个丢包率达到百分之一的糟糕网络，连续两次丢包的概率只有万分之一，这个概率实在是太小了，忽略它比解决它更具性价比。

因此，`TIME_WAIT` 和 `FIN_WAIT2` 状态的最大时长都是 2 MSL，由于在 Linux 系统中，MSL 的值固定为 30 秒，所以它们都是 60 秒。

虽然 `TIME_WAIT` 状态的存在是有必要的，但它毕竟在消耗系统资源，比如 `TIME_WAIT` 状态的端口就无法供新连接使用。怎样解决这个问题呢？

Linux 提供了 `tcp_max_tw_buckets` 参数，当 `TIME_WAIT` 的连接数量超过该参数时，新关闭的连接就不再经历 `TIME_WAIT` 而直接关闭。

 复制代码

```
1 net.ipv4.tcp_max_tw_buckets = 5000
```

当服务器的并发连接增多时，相应地，同时处于 TIME_WAIT 状态的连接数量也会变多，此时就应当调大 tcp_max_tw_buckets 参数，减少不同连接间数据错乱的概率。

当然，tcp_max_tw_buckets 也不是越大越好，毕竟内存和端口号都是有限的。有没有办法让新连接复用 TIME_WAIT 状态的端口呢？如果服务器会主动向上游服务器发起连接的话，就可以把 tcp_tw_reuse 参数设置为 1，它允许作为客户端的新连接，在安全条件下使用 TIME_WAIT 状态下的端口。

```
1 net.ipv4.tcp_tw_reuse = 1
```

[复制代码](#)

当然，要想使 tcp_tw_reuse 生效，还得把 timestamps 参数设置为 1，它满足安全复用的先决条件（对方也要打开 tcp_timestamps）：

```
1 net.ipv4.tcp_timestamps = 1
```

[复制代码](#)

老版本的 Linux 还提供了 tcp_tw_recycle 参数，它并不要求 TIME_WAIT 状态存在 60 秒，很容易导致数据错乱，不建议设置为 1。

```
1 net.ipv4.tcp_tw_recycle = 0
```

[复制代码](#)

所以在 Linux 4.12 版本后，直接取消了这一参数。

被动方的优化

当被动方收到 FIN 报文时，就开启了被动方的四次挥手流程。内核自动回复 ACK 报文后，连接就进入 CLOSE_WAIT 状态，顾名思义，它表示等待进程调用 close 函数关闭连接。

内核没有权力替代进程去关闭连接，因为若主动方是通过 shutdown 关闭连接，那么它就是想在半关闭连接上接收数据。**因此，Linux 并没有限制 CLOSE_WAIT 状态的持续时间。**

当然，大多数应用程序并不使用 shutdown 函数关闭连接，所以，当你用 netstat 命令发现大量 CLOSE_WAIT 状态时，要么是程序出现了 Bug，read 函数返回 0 时忘记调用 close 函数关闭连接，要么就是程序负载太高，close 函数所在的回调函数被延迟执行了。此时，我们应当在应用代码层面解决问题。

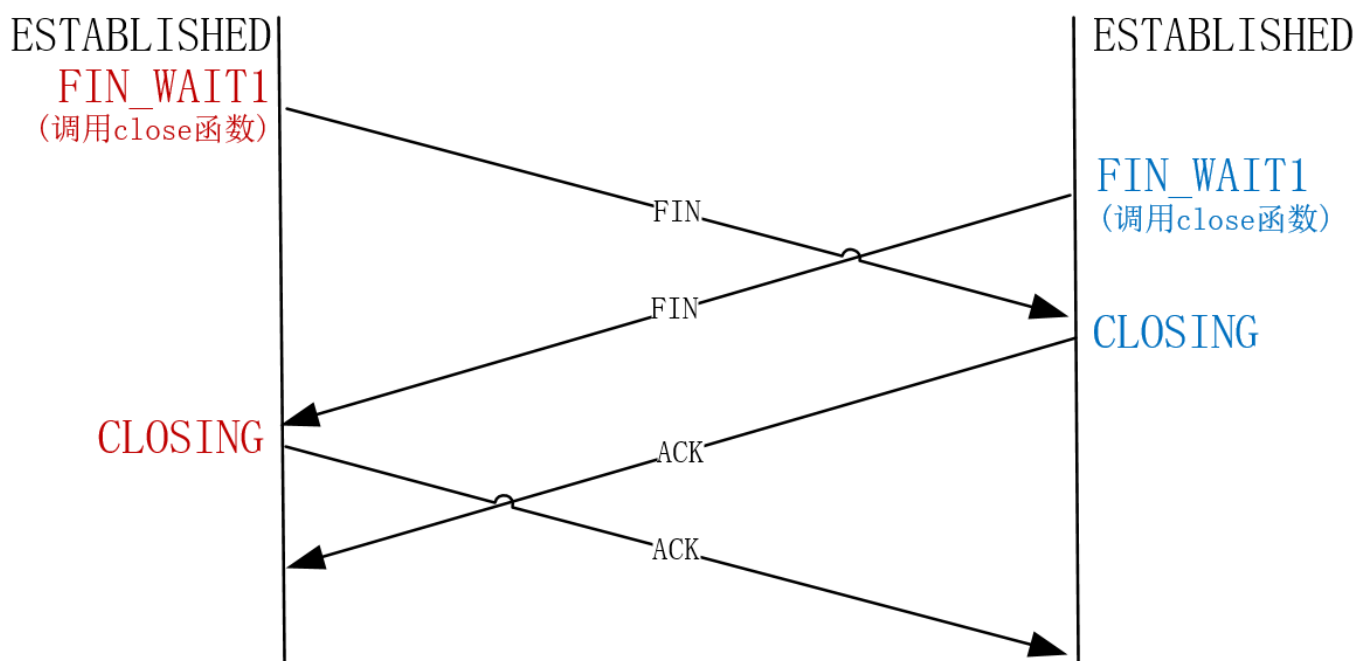
由于 CLOSE_WAIT 状态下，连接已经处于半关闭状态，所以此时进程若要关闭连接，只能调用 close 函数（再调用 shutdown 关闭单向通道就没有意义了），内核就会发出 FIN 报文关闭发送通道，同时连接进入 LAST_ACK 状态，等待主动方返回 ACK 来确认连接关闭。

如果迟迟等不到 ACK，内核就会重发 FIN 报文，重发次数仍然由 tcp_orphan_retries 参数控制，这与主动方重发 FIN 报文的优化策略一致。

至此，由一方主动发起四次挥手的流程就介绍完了。需要你注意的是，**如果被动方迅速调用 close 函数，那么被动方的 ACK 和 FIN 有可能在一个报文中发送，这样看起来，四次挥手会变成三次挥手，这只是一特殊情况，不用在意。**

我们再来看一种特例，如果连接双方同时关闭连接，会怎么样？

此时，上面介绍过的优化策略仍然适用。两方发送 FIN 报文时，都认为自己是主动方，所以都进入了 FIN_WAIT1 状态，FIN 报文的重发次数仍由 tcp_orphan_retries 参数控制。



接下来，双方在等待 ACK 报文的过程中，都等来了 FIN 报文。这是一种新情况，所以连接会进入一种叫做 CLOSING 的新状态，它替代了 FIN_WAIT2 状态。此时，内核回复 ACK 确认对方发送通道的关闭，仅己方的 FIN 报文对应的 ACK 还没有收到。所以，CLOSING 状态与 LAST_ACK 状态下的连接很相似，它会在适时重发 FIN 报文的情况下最终关闭。

小结

我们对这一讲的内容做个小结。

今天我们讲了四次挥手的流程，你需要根据主动方与被动方的连接状态变化来调整系统参数，使它在特定网络条件下更及时地释放资源。

四次挥手的主动方，为了应对丢包，允许在 `tcp_orphan_retries` 次数内重发 FIN 报文。当收到 ACK 报文，连接就进入了 FIN_WAIT2 状态，此时系统的行为依赖这是否为孤儿连接。

如果这是 `close` 函数关闭的孤儿连接，那么在 `tcp_fin_timeout` 秒内没有收到对方的 FIN 报文，连接就直接关闭，反之 `shutdown` 函数关闭的连接则不受此限制。毕竟孤儿连接可能在重发次数内存在数分钟之久，为了应对孤儿连接占用太多的资源，`tcp_max_orphans` 定义了最大孤儿连接的数量，超过时连接就会直接释放。

当接收到 FIN 报文，并返回 ACK 后，主动方的连接进入 TIME_WAIT 状态。这一状态会持续 1 分钟，为了防止 TIME_WAIT 状态占用太多的资源，`tcp_max_tw_buckets` 定义了最大数量，超过时连接也会直接释放。当 TIME_WAIT 状态过多时，还可以通过设置 `tcp_tw_reuse` 和 `tcp_timestamps` 为 1，将 TIME_WAIT 状态的端口复用于作为客户端的新连接。

被动关闭的连接方应对非常简单，它在回复 ACK 后就进入了 CLOSE_WAIT 状态，等待进程调用 `close` 函数关闭连接。因此，出现大量 CLOSE_WAIT 状态的连接时，应当从应用程序中找问题。当被动方发送 FIN 报文后，连接就进入 LAST_ACK 状态，在未等来 ACK 时，会在 `tcp_orphan_retries` 参数的控制下重发 FIN 报文。

至此，TCP 连接建立、关闭时的性能优化就介绍完了。下一讲，我们将专注在 TCP 上传输数据时，如何优化内存的使用效率。

思考题

最后，给你留一个思考题。你知道关闭连接时的 `SO_LINGER` 选项吗？它希望用四次挥手替代 `RST` 关闭连接的方式，防止浏览器没有接收到完整的 HTTP 响应。请你思考一下，`SO_LINGER` 会怎么影响主动方连接的状态变化？`SO_LINGER` 上的超时时间，是怎样与系统配置参数协作的？欢迎你在留言区与我一起探讨。

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。

课程预告

6月-7月课表抢先看

充 ¥500 得 ¥580

赠「¥ 118 月球主题 AR 笔记本」



【点击】图片, 立即查看>>>

© 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 09 | 如何提升TCP三次握手的性能？

下一篇 11 | 如何修改TCP缓冲区才能兼顾并发数量与传输速度？

精选留言 (12)

写留言



忆水寒

2020-05-21

本节内容真的很烧脑啊！收获也很多，从来没有从这么多角度看待过这个问题。

so_linger是一个结构体，其中有两个参数：l_onoff和l_linger。第一个参数表示是否启用so_linger功能，第二个参数l_linger=0，则close函数在阻塞直到l_linger时间超时或者数据发送完毕，如果超时则直接情况缓冲区然后RST连接（默认情况下是调用close函数立即返回）。

展开 ∨

作者回复: 谢谢忆水寒的分享！

2 5



东郭

2020-05-21

看到一遍文章，tcp_timestamps还有这个坑吗？

<https://blog.51cto.com/fuyuan2016/1795998>

作者回复: 你好东郭，我认为本质上这是打开tcp_tw_recycle造成的，真不关timestamps的事。timestamps的好处很多，不建议关掉。

1 1



流浪地球

2020-05-20

MSL的数值在所有网络环境中都一样，如果一些跨洋的长距离网络这个时间可以保证吗？如果发现这个数值无法保证，是需要调整网络传输层次结构吗？

展开 ∨

1 1



我来也

2020-05-20

老师的文章都要细品。

内核调优的参数算是见识到了,不知道啥时候才能用得上.🙏

一些边界情况老师也介绍到了:

1. tcp_tw_reuse 也是有适用条件的....

展开 ∨

作者回复: Nginx中有SO_LINGER选项的应用，可以参见《Nginx核心知识100讲》第130课



1

**安排**

2020-05-20

TTL每一跳减少1，这些怎么和MSL对应起来呢，每一跳减少的1相当于1秒？

作者回复: 不是，这是一个预估值，所谓每一跳，是指每经过一个路由器网络设备，将IP头部中的TTL字段减少1，并不等于1秒，通常推荐的TTL的初始值是64



1

**Geek_David**

2020-05-29

一篇15分钟的音频，足足花了1.5个小时
包括先听音频，再看文字，再理解每个含义，再记笔记
我敢说我懂了。
前提是对TCP想过比较了解的情况下，太南了

**Cola**

2020-05-24

有个问题请教一下，关于so_linger选项，在一个网络框架服务端源码上看到针对监听socket，会设置l_onff=1, l_linger=0，针对accept之后建立的连接socket，会设置l_onff=0, l_linger=0，这里的用意是什么呢，l_onff=1, l_linger=0可以避免TW状态，但是在监听socket上如此设置还是这个目的吗？

展开 ▾

**那一刻**

2020-05-20

请问一个关于rst的问题，上篇文章提到过accept队列满的时候，会发rst。这里提到 tcp_max_orphans 参数，如果孤儿连接数量大于它，新增的孤儿连接将不再走四次挥手，而是直接发送 RST 复位报文强制关闭。我们在发送http请求时，会遇到rest by peer，是对应这个消息么？另外怎么区分是由于accept队列满还是由于孤儿进程数量多，导致rst呢？

展开 ▾

2

**唐朝首都**

2020-05-20

(1) 问题1：FIN_WAIT1状态的连接数较多

原因1：因长时间没有收到被动方的ACK：重发次数控制tcp_orphan_retries，降低重发次数。

原因2：FIN信息发不出去

调整tcp_max_orphans参数，当孤儿连接数大于它时，新增的孤儿连接数不在走四次挥...

展开 ▾



@Hy

2020-05-20

当然，大多数应用程序并不使用 shutdown 函数关闭连接，所以，当你用 netstat 命令发现大量 CLOSE_WAIT 状态时，要么是程序出现了 Bug，read 函数返回 0 时忘记调用 close 函数关闭连接，要么就是程序负载太高，close 函数所在的回调函数被延迟执行了。此时，我们应当在应用代码层面解决问题。

...

展开 ▾



夜空中最亮的星 (华仔...)

2020-05-20

至少要听3遍

展开 ▾

作者回复: 点个赞!



安排

2020-05-20

rst报文的发送是比普通数据优先级高吗，也就是socket对端如果先接受到rst，这个socket就不可读不可写了，后面收到的数据自然也没法被进程拿到了。

展开 ▾

