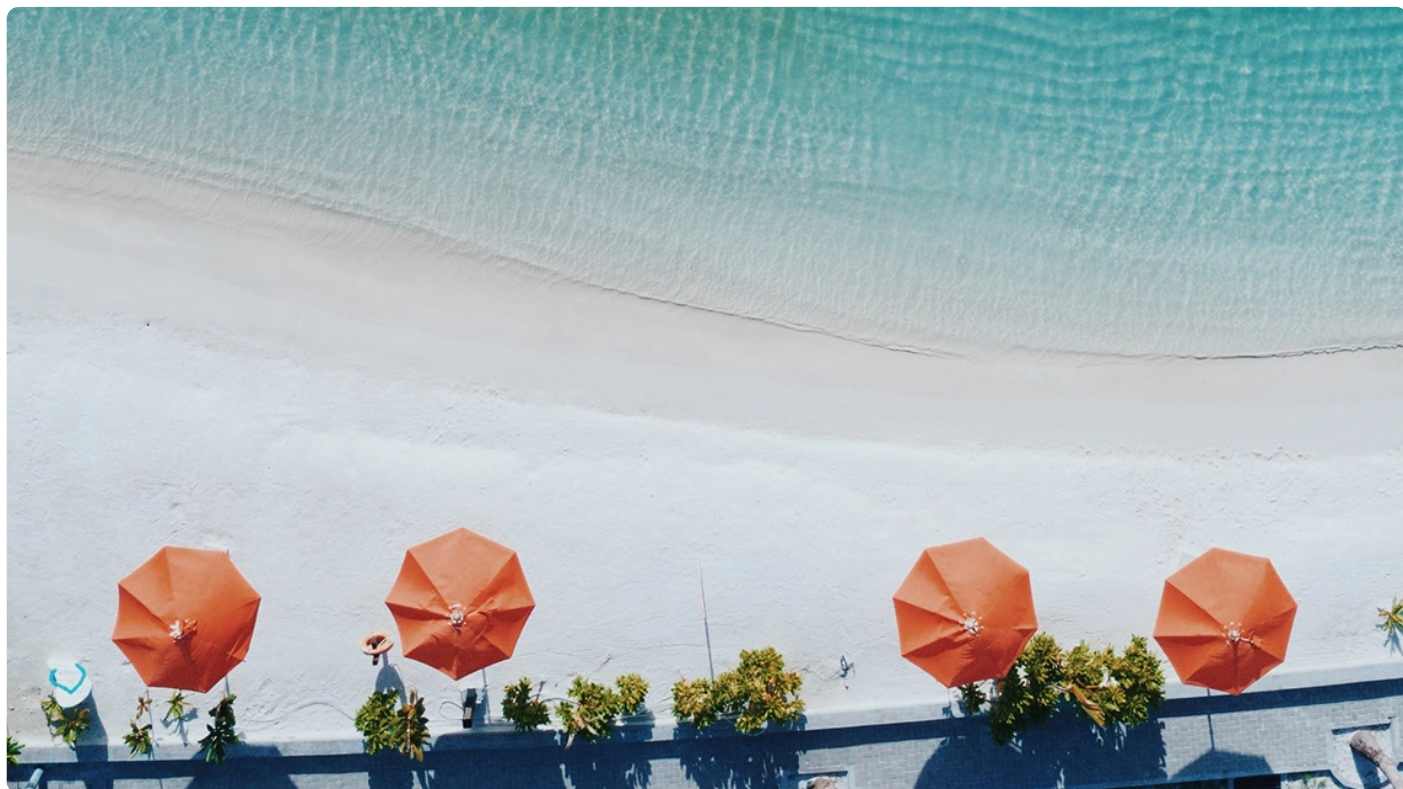


# 10 | TIME\_WAIT：隐藏在细节下的魔鬼

2019-08-23 盛延敏 来自北京

《网络编程实战》



你好，我是盛延敏，这是网络编程实战的第 10 讲，欢迎回来。

在前面的基础篇里，我们对网络编程涉及到的基础知识进行了梳理，主要内容包括 C/S 编程模型、TCP 协议、UDP 协议和本地套接字等内容。在提高篇里，我将结合我的经验，引导你对 TCP 和 UDP 进行更深入的理解。

学习完提高篇之后，我希望你对如何提高 TCP 及 UDP 程序的健壮性有一个全面清晰的认识，从而为深入理解性能篇打下良好的基础。

在前面的基础篇里，我们了解了 TCP 四次挥手，在四次挥手的过程中，发起连接断开的一方会有一段时间处于 TIME\_WAIT 的状态，你知道 TIME\_WAIT 是用来做什么的么？在面试和实战中，TIME\_WAIT 相关的问题始终是绕不过去的一道难题。下面就请跟随我，一起找出隐藏在细节下的魔鬼吧。

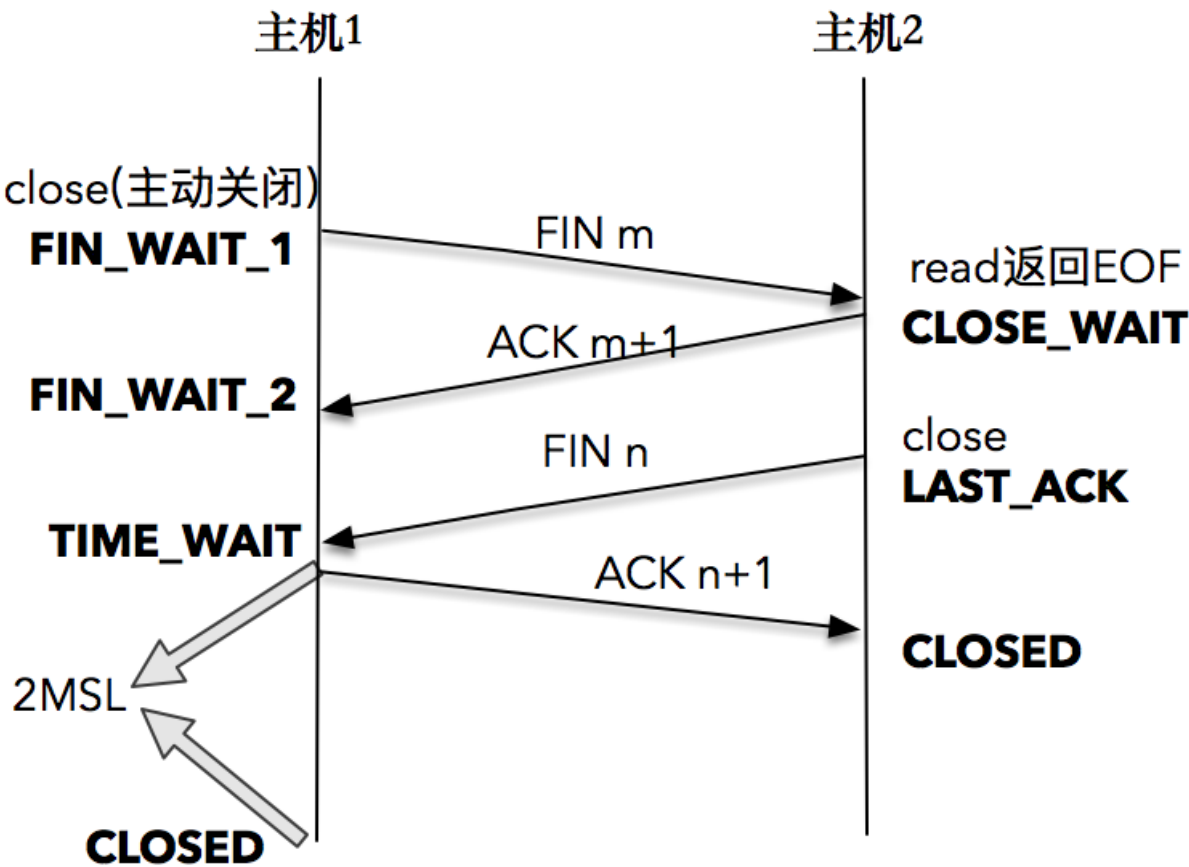
# TIME\_WAIT 发生的场景

让我们先从一例线上故障说起。在一次升级线上应用服务之后，我们发现该服务的可用性变得时好时坏，一段时间可以对外提供服务，一段时间突然又不可以，大家都百思不得其解。运维同学登录到服务所在的主机上，使用 netstat 命令查看后才发现，主机上有成千上万处于 TIME\_WAIT 状态的连接。

经过层层剖析后，我们发现罪魁祸首就是 TIME\_WAIT。为什么呢？我们这个应用服务需要通过发起 TCP 连接对外提供服务。每个连接会占用一个本地端口，当在高并发的情况下，TIME\_WAIT 状态的连接过多，多到把本机可用的端口耗尽，应用服务对外表现的症状，就是不能正常工作了。当过了一段时间之后，处于 TIME\_WAIT 的连接被系统回收并关闭后，释放出本地端口可供使用，应用服务对外表现为，可以正常工作。这样周而复始，便会出现了一会儿不可以，过一两分钟又可以正常工作的现象。

那么为什么会产生这么多的 TIME\_WAIT 连接呢？

这要从 TCP 的四次挥手说起。



TCP 连接终止时，主机 1 先发送 FIN 报文，主机 2 进入 CLOSE\_WAIT 状态，并发送一个 ACK 应答，同时，主机 2 通过 read 调用获得 EOF，并将此结果通知应用程序进行主动关闭操作，发送 FIN 报文。主机 1 在接收到 FIN 报文后发送 ACK 应答，此时主机 1 进入 TIME\_WAIT 状态。

主机 1 在 TIME\_WAIT 停留持续时间是固定的，是最长分节生命期 MSL (maximum segment lifetime) 的两倍，一般称之为 2MSL。和大多数 BSD 派生的系统一样，Linux 系统里有一个硬编码的字段，名称为 TCP\_TIMEWAIT\_LEN，其值为 60 秒。也就是说，**Linux 系统停留在 TIME\_WAIT 的时间为固定的 60 秒。**

 复制代码

```
1 #define TCP_TIMEWAIT_LEN (60*HZ) /* how long to wait to destroy TIME-
```

```
WAIT
```

过了这个时间之后，主机 1 就进入 CLOSED 状态。为什么是这个时间呢？你可以先想一想，稍后我会给出解答。

你一定要记住一点，**只有发起连接终止的一方会进入 TIME\_WAIT 状态。**这一点面试的时候经常会被问到。

## TIME\_WAIT 的作用

你可能会问，为什么不直接进入 CLOSED 状态，而要停留在 TIME\_WAIT 这个状态？

这要从两个方面来说。

首先，这样做是为了确保最后的 ACK 能让被动关闭方接收，从而帮助其正常关闭。

TCP 在设计的时候，做了充分的容错性设计，比如，TCP 假设报文会出错，需要重传。在这里，如果图中主机 1 的 ACK 报文没有传输成功，那么主机 2 就会重新发送 FIN 报文。

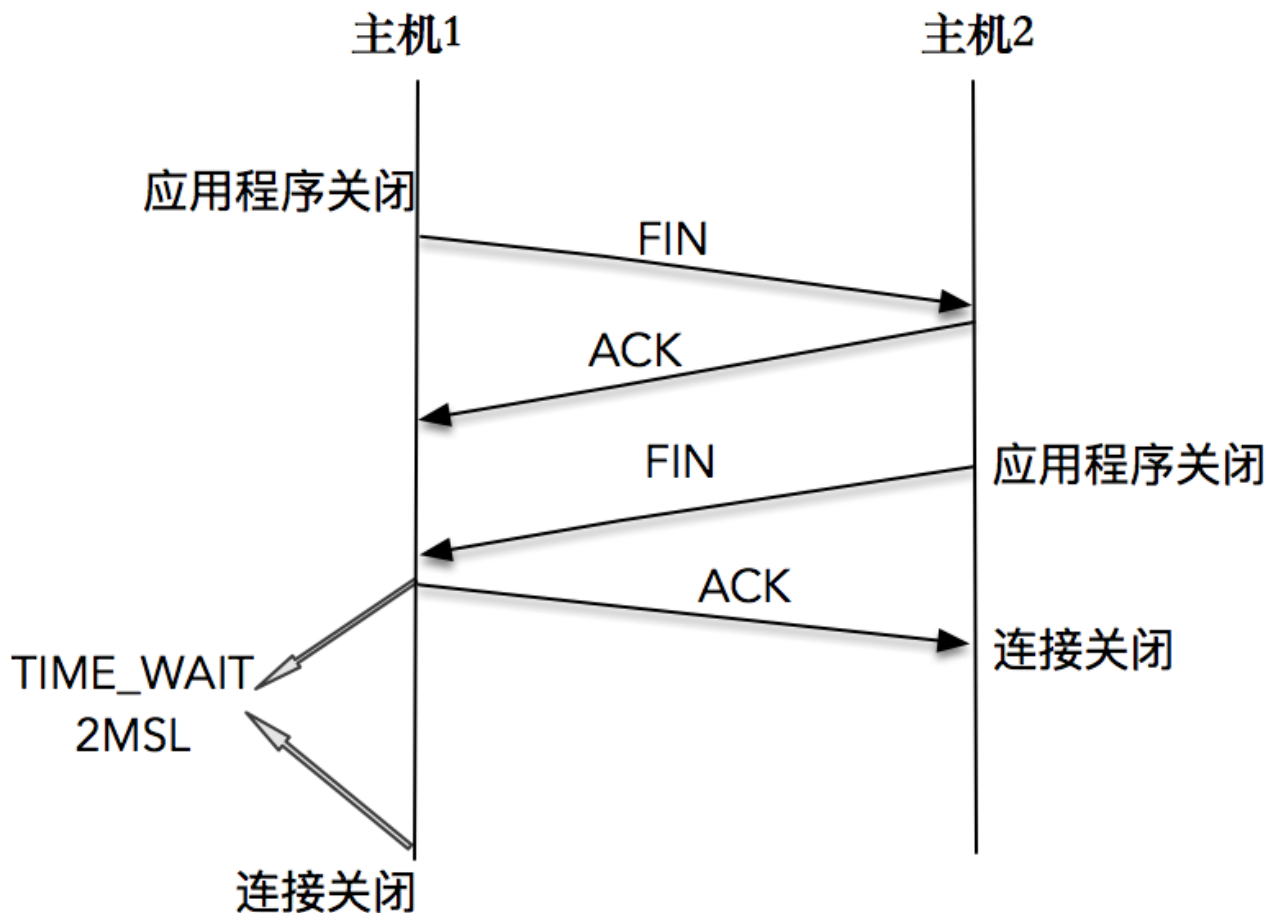
如果主机 1 没有维护 TIME\_WAIT 状态，而直接进入 CLOSED 状态，它就失去了当前状态的上下文，只能回复一个 RST 操作，从而导致被动关闭方出现错误。

现在主机 1 知道自己处于 TIME\_WAIT 的状态，就可以在接收到 FIN 报文之后，重新发出一个 ACK 报文，使得主机 2 可以进入正常的 CLOSED 状态。

第二个理由和连接“化身”和报文迷走有关系，为了让旧连接的重复分节在网络中自然消失。

我们知道，在网络中，经常会发生报文经过一段时间才能到达目的地的情况，产生的原因是多种多样的，如路由器重启，链路突然出现故障等。如果迷走报文到达时，发现 TCP 连接四元组（源 IP，源端口，目的 IP，目的端口）所代表的连接不复存在，那么很简单，这个报文自然丢弃。

我们考虑这样一个场景，在原连接中断后，又重新创建了一个原连接的“化身”，说是化身其实是因为这个连接和原先的连接四元组完全相同，如果迷失报文经过一段时间也到达，那么这个报文会被误认为是连接“化身”的一个 TCP 分节，这样就会对 TCP 通信产生影响。



所以，TCP 就设计出了这么一个机制，经过 2MSL 这个时间，足以让两个方向上的分组都被丢弃，使得原来连接的分组在网络中都自然消失，再出现的分组一定都是新化身所产生的。

划重点，2MSL 的时间是**从主机 1 接收到 FIN 后发送 ACK 开始计时的**；如果在 TIME\_WAIT 时间内，因为主机 1 的 ACK 没有传输到主机 2，主机 1 又接收到了主机 2 重发的 FIN 报文，那么 2MSL 时间将重新计时。道理很简单，因为 2MSL 的时间，目的是为了旧连接的所有报文都能自然消亡，现在主机 1 重新发送了 ACK 报文，自然需要重新计时，以便防止这个 ACK 报文对新可能的连接化身造成干扰。

## TIME\_WAIT 的危害

过多的 TIME\_WAIT 的主要危害有两种。

第一是内存资源占用，这个目前看来不是太严重，基本可以忽略。

第二是对端口资源的占用，一个 TCP 连接至少消耗一个本地端口。要知道，端口资源也是有限的，一般可以开启的端口为 32768 ~ 61000，也可以通过 `net.ipv4.ip_local_port_range` 指定，如果 TIME\_WAIT 状态过多，会导致无法创建新连接。这个也是我们在一开始讲到的那个例子。

## 如何优化 TIME\_WAIT?

在高并发的情况下，如果我们想对 TIME\_WAIT 做一些优化，来解决我们一开始提到的例子，该如何办呢？

### `net.ipv4.tcp_max_tw_buckets`

一个暴力的方法是通过 `sysctl` 命令，将系统值调小。这个值默认为 18000，当系统中处于 TIME\_WAIT 的连接一旦超过这个值时，系统就会将所有的 TIME\_WAIT 连接状态重置，并且只打印出警告信息。这个方法过于暴力，而且治标不治本，带来的问题远比解决的问题多，不推荐使用。

### 调低 TCP\_TIMEWAIT\_LEN，重新编译系统


这个方法是一个不错的方法，缺点是需要“一点”内核方面的知识，能够重新编译内核。我想这个不是大多数人能接受的方式。

## SO\_LINGER 的设置

英文单词“linger”的意思为停留，我们可以通过设置套接字选项，来设置调用 close 或者 shutdown 关闭连接时的行为。

 复制代码

```
1 int setsockopt(int sockfd, int level, int optname, const void *optval,  
2               socklen_t optlen);
```


 复制代码

```
1 struct linger {  
2     int    l_onoff;        /* 0=off, nonzero=on */  
3     int    l_linger;       /* linger time, POSIX specifies units as seconds */  
4 }
```

设置 linger 参数有几种可能：

如果 `l_onoff` 为 0，那么关闭本选项。`l_linger` 的值被忽略，这对应了默认行为，close 或 shutdown 立即返回。如果在套接字发送缓冲区中有数据残留，系统会将试着把这些数据发送出去。

如果 `l_onoff` 为非 0，且 `l_linger` 值也为 0，那么调用 close 后，会立刻发送一个 RST 标志给对端，该 TCP 连接将跳过四次挥手，也就跳过了 TIME\_WAIT 状态，直接关闭。这种关闭的方式称为“强行关闭”。在这种情况下，排队数据不会被发送，被动关闭方也不知道对端已经彻底断开。只有当被动关闭方正阻塞在 `recv()` 调用上时，接受到 RST 时，会立刻得到一个“connection reset by peer”的异常。

 复制代码

```
1 struct linger so_linger;  
2 so_linger.l_onoff = 1;  
3 so_linger.l_linger = 0;  
4 setsockopt(s, SOL_SOCKET, SO_LINGER, &so_linger, sizeof(so_linger));
```



如果`l_onoff`为非 0，且`l_linger`的值也非 0，那么调用 `close` 后，调用 `close` 的线程就将阻塞，直到数据被发送出去，或者设置的`l_linger`计时时间到。


第二种可能为跨越 `TIME_WAIT` 状态提供了一个可能，不过是一个非常危险的行为，不值得提倡。

## **net.ipv4.tcp\_tw\_reuse：更安全的设置**

那么 Linux 有没有提供更安全的选择呢？

当然有。这就是`net.ipv4.tcp_tw_reuse`选项。

Linux 系统对于`net.ipv4.tcp_tw_reuse`的解释如下：

 复制代码

```
1 Allow to reuse TIME-WAIT sockets for new connections when it is safe from protoco
```

这段话的大意是从协议角度理解如果是安全可控的，可以复用处于 `TIME_WAIT` 的套接字为新的连接所用。

那么什么是协议角度理解的安全可控呢？主要有两点：

1. 只适用于连接发起方（C/S 模型中的客户端）；
2. 对应的 `TIME_WAIT` 状态的连接创建时间超过 1 秒才可以被复用。

使用这个选项，还有一个前提，需要打开对 TCP 时间戳的支持，即`net.ipv4.tcp_timestamps=1`（默认即为 1）。

要知道，TCP 协议也在与时俱进，RFC 1323 中实现了 TCP 拓展规范，以便保证 TCP 的高可用，并引入了新的 TCP 选项，两个 4 字节的时间戳字段，用于记录 TCP 发送方的当前时间戳

和从对端接收到的最新时间戳。由于引入了时间戳，我们在前面提到的 2MSL 问题就不复存在了，因为重复的数据包会因为时间戳过期被自然丢弃。

## 总结

在今天的内容里，我讲了 TCP 的四次挥手，重点对 TIME\_WAIT 的产生、作用以及优化进行了讲解，你需要记住以下三点：

TIME\_WAIT 的引入是为了让 TCP 报文得以自然消失，同时为了让被动关闭方能够正常关闭；

不要试图使用 SO\_LINGER 设置套接字选项，跳过 TIME\_WAIT；

现代 Linux 系统引入了更安全可控的方案，可以帮助我们尽可能地复用 TIME\_WAIT 状态的连接。

## 思考题

最后按照惯例，我留两道思考题，供你消化今天的内容。

1. 最大分组 MSL 是 TCP 分组在网络中存活的最长时间，你知道这个最长时间是如何达成的？换句话说，是怎么样机制，可以保证在 MSL 达到之后，报文就自然消亡了呢？
2. RFC 1323 引入了 TCP 时间戳，那么这需要在发送方和接收方之间定义一个统一的时钟吗？

欢迎你在评论区写下你的思考，如果通过这篇文章你理解了 TIME\_WAIT，欢迎你把这篇文章分享给你的朋友或者同事，一起交流学习一下。

© 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。

## 精选留言 (76)



Leon 置顶

2019-08-23



net.ipv4.tcp\_tw\_recycle是客户端和服务端都可以复用，但是容易造成端口接收数据混乱，4.12内核直接砍掉了，老师是因为内核去掉了所以没提了嘛

作者回复: BINGO。太多了，大家也不好接受，我想我还是不求面面俱到，但求有所启发和引领。

共 2 条评论 >

👍 29



**何某人**

2019-08-23

老师，那么通过setsockopt设置SO\_REUSEADDR这个方法呢？网上资料基本上都是通过设置这个来解决TIME\_WAIT。这个方法有什么优劣吗？

作者回复: 这个是解决端口复用的问题，并不是解决TIME\_WAIT，这个是告诉内核，即使TIME\_WAIT状态的套接字，我也可以继续使用它做为新的套集字使用。

共 6 条评论 >

👍 32



**雷神的盛宴**

2019-12-20

net.ipv4.tcp\_tw\_reuse 要慎用，当客户端与服务端主机时间不同步时，客户端的发送的消息会被直接拒绝掉

作者回复: 学到了

共 6 条评论 >

👍 27



**许童童**

2019-08-23

1. MSL的意思是最长报文段寿命。IP头部中有个TTL字段意思是生存时间。TTL每经过一个路由器就减1，到0就会被丢弃，而MSL是由RFC里面规定的2分钟，但实际在工程上2分钟太长，因此TCP允许根据具体的情况配置大小，TTL与MSL是有关系的但不是简单的相等关系，MSL要大于TTL。MSL内部应该就是一个普通的定时器实现的。
- 2.不需要统一时钟，可以在第一次交换双方的时钟，之后用相对时间就可以了。

共 1 条评论 >

👍 23



**AnonymousUser**

2019-09-04

TIME\_WAIT的作用：

- 1) 确保对方能够正确收到最后的ACK，帮助其关闭；
- 2) 防迷走报文对程序带来的影响。

TIME\_WAIT的危害：

- 1) 占用内存；
- 2) 占用端口。

作者回复：总结很到位。

共 2 条评论 >

👍 20



**张天屹**

2020-03-30

文中的问题有个前提，必须是监听的端口。我看有评论提到80,8080这种服务，是否只能同时访问一次？答案是否定的。因为网络中的服务分为监听端口和连接端口，当建立一个连接之后，监听端口是不被占用的，此时会用一个新的端口来建立连接，而且就算是新的端口，一个TCP连接也是（客户端ip,客户端端口，服务端ip，服务端端口）共同决定的，不冲突。

作者回复：帮我回答了，赞

共 4 条评论 >

👍 14



**alan**

2019-08-23

TIME\_WAIT的作用：

1. 确保主动断开方的最后一个ACK成功发到对方
2. 确保残留的TCP包自然消亡

共 1 条评论 >

👍 10



**丹枫无迹**

2020-02-02

由于引入了时间戳，我们在前面提到的 2MSL 问题就不复存在了，因为重复的数据包会因为时间戳过期被自然丢弃。

这个没理解，为什么 2MSL 问题就不存在了？老师能解释下吗？

作者回复：这是因为时间戳会告诉我们报文发送的时间，这样在迷走报文和正确报文同时到达的情况下，我们就可以很方便的分辩出应该丢弃掉那个报文，并不会对最后收到的报文产生任何不利的影响。

共 2 条评论 >

👍 8



**云师兄**

2019-12-13

Reuse只适用于连接发起方（C/S 模型中的客户端），但目前要解决的是服务端连接不足问题，这个方法要如何发挥作用呢？

作者回复: 这里针对的TIME\_WAIT是指主动关闭的一方，不一定是客户端或者服务器端，如果服务器端主动关闭连接，也是属于这样的范畴的。



👍 9



**吴光庆**

2019-08-26

为什么是2MSL，不是3 MSL，4 MSL。

作者回复: 因为是一来一返。

共 2 条评论 >

👍 9



**JasonZhi**

2019-09-10

SO\_REUSEADDR和SO\_REUSEPORT可以详细说下作用吗？有点迷糊，文章都没有说明这两个参数，评论区就冒出一大堆关于这两个参数的评论。

作者回复: 哈哈，提前剧透啊。

这是为了解决如何快速复用处于TIME\_WAIT的连接，如果不设置这个选项，处于TIME\_WAIT的连接是不能被快速复用的，必须等待系统回收连接才可以，如果这个时候开启服务器端口，会报地址已被占用的错误。

这块在第15讲里会有详细阐述。



👍 7



**pc**

2020-05-25

先说点其他的吧：看完了基础篇，收获了很多，也更加期待后面的内容（本来就是冲着time\_wait和epoll来到这个课程）。当然其中也遇到很多问题，其中也在评论区提了两个。本来以为这么长时间了老师也不会再回复了，周末一看，竟然回复了我的问题！其实一边是开心，另一边是得到答案的开心（其实自己也搜索过，但是感觉搜到文章都不是我想问的内容）。

### 【提问啦～】

- 1、看到评论区的“通过setsockopt设置SO\_REUSEADDR这个方法”，感觉和net.ipv4.tcp\_tw\_reuse选项的作用也很像，都是端口复用，只是后者是在安全可控的基础上---这样理解对吗？
- 2、老师在文中说的“过了2MSL这个时间之后，主机 1 就进入 CLOSED 状态”，我自己还是没有总结出答案，是评论区所说的“去时ACK的最大存活时间（MSL）+来时FIIN的最大存活时间（MSL）= 2MSL”这个原因吗？
- 3、TIME\_WAIT=2MSL和TCP\_TIMEWAIT\_LEN有啥关系？是：TIME\_WAIT实际上是由TCP\_TIMEWAIT\_LEN控制，然后只不过其值约等于2MSL来控制迷走报文的消亡 这样么？
- 4、文中说TCP\_TIMEWAIT\_LEN、net.ipv4.tcp\_tw\_reuse都是linux的选项，但是客户端来说的话，有android、ios、windows各种系统吧？是每个系统都有类似的控制选项么？

作者回复：我一直都在回复的哦，和大家在一起自己也学到不少，哈哈。

以下试着回答你的问题：

- 1、看到评论区的“通过setsockopt设置SO\_REUSEADDR这个方法”，感觉和net.ipv4.tcp\_tw\_reuse选项的作用也很像，都是端口复用，只是后者是在安全可控的基础上---这样理解对吗？

我认为不对哦，前者是针对服务端连接地址被占用的情况，后者是针对连接发起方；

- 2、老师在文中说的“过了2MSL这个时间之后，主机 1 就进入 CLOSED 状态”，我自己还是没有总结出答案，是评论区所说的“去时ACK的最大存活时间（MSL）+来时FIIN的最大存活时间（MSL）= 2MSL”这个原因吗？

你的意思是为什么要制定2MSL这个时间段才CLOSED是么？如果是这样，评论去的“去时ACK的最大存活时间（MSL）+来时FIIN的最大存活时间（MSL）= 2MSL”算是一个靠谱的理解吧。

- 3、TIME\_WAIT=2MSL和TCP\_TIMEWAIT\_LEN有啥关系？是：TIME\_WAIT实际上是由TCP\_TIMEWAIT\_LEN控制，然后只不过其值约等于2MSL来控制迷走报文的消亡 这样么？

TIME\_WAIT是一个抽象的定义，而TCP\_TIMEWAIT\_LEN是Linux默认的值，是一个常量。你的认识是对的

4、文中说TCP\_TIMEWAIT\_LEN、net.ipv4.tcp\_tw\_reuse都是linux的选项，但是客户端来说的话，有android、ios、windows各种系统吧？是每个系统都有类似的控制选项么？

Android是裁剪过的Linux，应该可以复用；其他两个我不是很清楚，想必应该有自己的控制选项。



👍 5



**tongmin\_tsai**

2019-09-30

老师，我有疑问的是，IP包中TTL每经过一次路由就少1，那么2MSL怎么确保可以一定大于TTL的？

作者回复：只要每次经过一跳的时间肯定大于1秒以上就可以了，实际处理的时间肯定大于这个值的。

2MSL设置的为60秒，TTL设置为60，只有每次一跳都大于1秒，那么肯定时间总和大于60秒了。

共 5 条评论 >

👍 5



**magicnum**

2019-09-01

- 1.记录一个值，比如60s，经过一个网关就减去一定短值，值=0的时候网关决定丢弃；
- 2.不需要。timestamp不需要交互，只是发送方使用的

共 3 条评论 >

👍 4



**Geek\_9a0180**

2019-08-23

印象中是当一端关闭socket连接，另一端如果尝试从TCP连接中读取数据，则会报connect reset，如果尝试向连接中写入数据，则会报connect reset by peer，好像和老师说的正相反，还请老师帮忙解答一下，谢谢：)

作者回复：首先，我不明白connect reset和connect reset by peer有啥区别哎，这两个不是一回事么？都是RST信号。

其次，我确认读的时候会读到FIN报文，连续写则会得到RST结果。

共 2 条评论 >

👍 4



**Dovelol**

2019-08-23

老师好，想问个问题，一般我们服务器上运行一个服务，比如tomcat，zk这种，然后监听8080或2181端口，这时候外部直连（不再经过web服务器转发）的话，虽然有很多连接但服务端应该都是只占用一个端口，也就是说netstat -anp命令看到的本机都是ip+固定端口，那么此时如果服务端主动关闭一些连接的话，也会有大量time\_wait问题对吧，但此时好像并没有消耗更多端口，那这个影响对于服务端来说是什么呢？老师讲的出现大量time\_wait应该都是在客户端的一方吧，因为客户端发起请求会占用一个新端口，主动关闭到time\_wait阶段就相当于这个新的端口一直被占用。我还有个疑问是，这种大量time\_wait在连接数多的情况下是肯定会出现的，是不是可以从减少连接的方向去解决问题呢，比如用连接池这种技术可以解决吗？

作者回复：这个时候你看到的连接都是服务器端被动建立的连接，本地端口都是服务器监听的端口，类似

<127.0.0.1, 80, ip1, 51231>

<127.0.0.1, 80, ip2, 51331>

...

所以，不会存在我讲到的那个问题，这些个连接过了一段时间自然会被回收。

连接池是为了多个线程复用连接，减少TCP连接的数量，是为了更高效的使用TCP，确实也客观减少了连接的数量。

共 4 条评论 >

👍 4



**列夫托尔斯泰克洛伊来...**

2020-07-08

这个可控优化的方法没明白，是复用端口的意思吗？不过复用端口数据不混乱了？

作者回复：不是端口复用，是复用处于 TIME\_WAIT 的套接字为新的连接所用。前提在文中也提到了，是通过TCP时间戳来解决2MSL的问题。



👍 3



**Liam**

2019-08-24

老师好，我又2个问题不明白：

1 为什么说time\_wait会占用过多端口，难道不是占用socket而已吗？比如一个server与多个client建立多个连接，对于server而言只会占用一个端口吧

2 什么是报文的自然消亡，指的是报文发送到对方或报文正常丢弃吗？然后对连接化身这段看不明白



作者回复: 1. 我说的情况是主动断开连接的一方, 比如一个客户端每次对外建立一个连接, 这是要消耗一个本地端口的, 只不过这个端口在我们建立连接的时候由内核帮我们选好了;

2. 报文的自然消亡, 就是TTL时间为0了, 不会在网络中继续传播, 到了某个网络设备, 报文会被丢弃掉。



**Initiative Thinker**

2021-08-24

复用后的套接字, 如何恢复旧连接的FIN呢?

作者回复: 是说"回复"吧?

处于 TIME\_WAIT 的套接字为新的连接所用, 通过时间戳可以知道旧连接的FIN是一个无效的FIN, 从而直接回复RST, 让旧连接直接出错退出。



**Geek\_9edd4f**

2021-03-02

"第二是对端口资源的占用, 一个 TCP 连接至少消耗一个本地端口。要知道, 端口资源也是有限的, 一般可以开启的端口为 32768 ~ 61000, 也可以通过net.ipv4.ip\_local\_port\_range指定, 如果 TIME\_WAIT 状态过多, 会导致无法创建新连接。这个也是我们在一开始讲到的那个例子。"

这里不是很理解, 服务端提供服务应该就只用一个端口号吧? 而客户端请求服务应该也是只使用一个端口号吧? 普通个人客户就发起一个请求只用一个端口号, 为什么会出现端口号不够用的情况? 难道指的为客户服务的代理服务器可能会端口号不够用吗? 因为代理服务器要处理来自成千上万的客户请求, 需要选择不同的端口号为客户服务, 将请求发给服务器吗?

作者回复: 这个例子是客户端发起连接过多, 每次发起连接都会占用一个端口。客户端和服务端是相对, 比如一个应用程序对于客户的请求是服务端, 同时为了服务这个客户请求, 又要向另一个服务发起调用请求(典型的例子是向数据库发起连接请求)。

