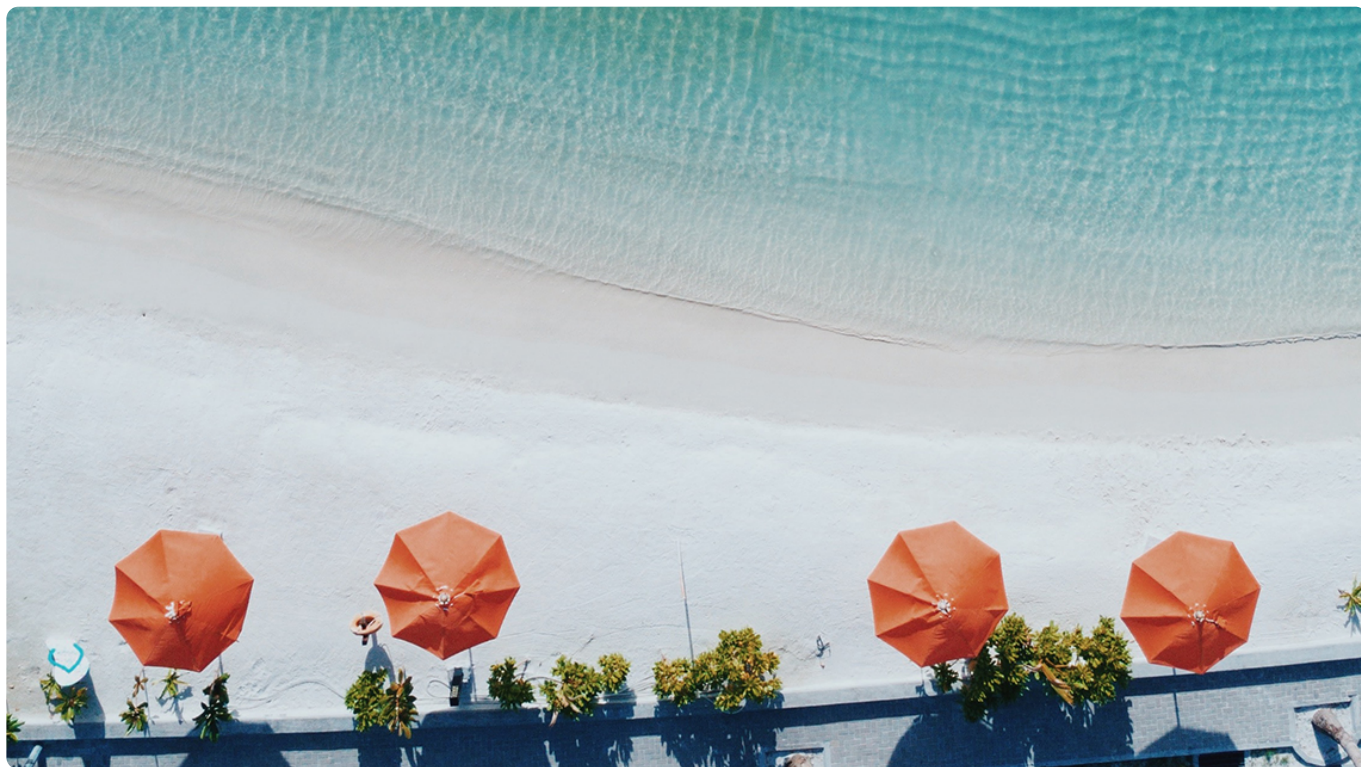


## 10 | TIME\_WAIT：隐藏在细节下的魔鬼

2019-08-23 盛延敏

网络编程实战

[进入课程 >](#)



讲述：冯永吉

时长 10:57 大小 10.04M



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。

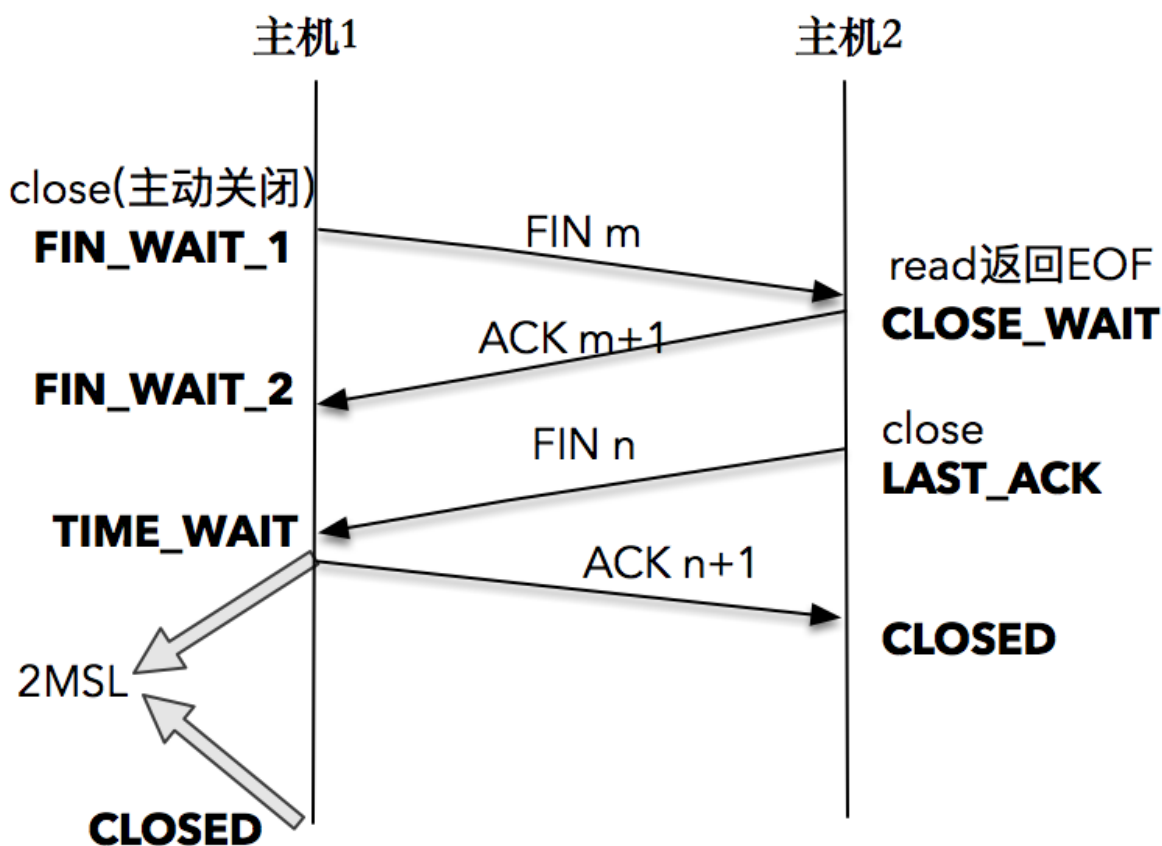
## TIME\_WAIT 发生的场景

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。我在文稿中放了这样一张图。



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 state
```

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。

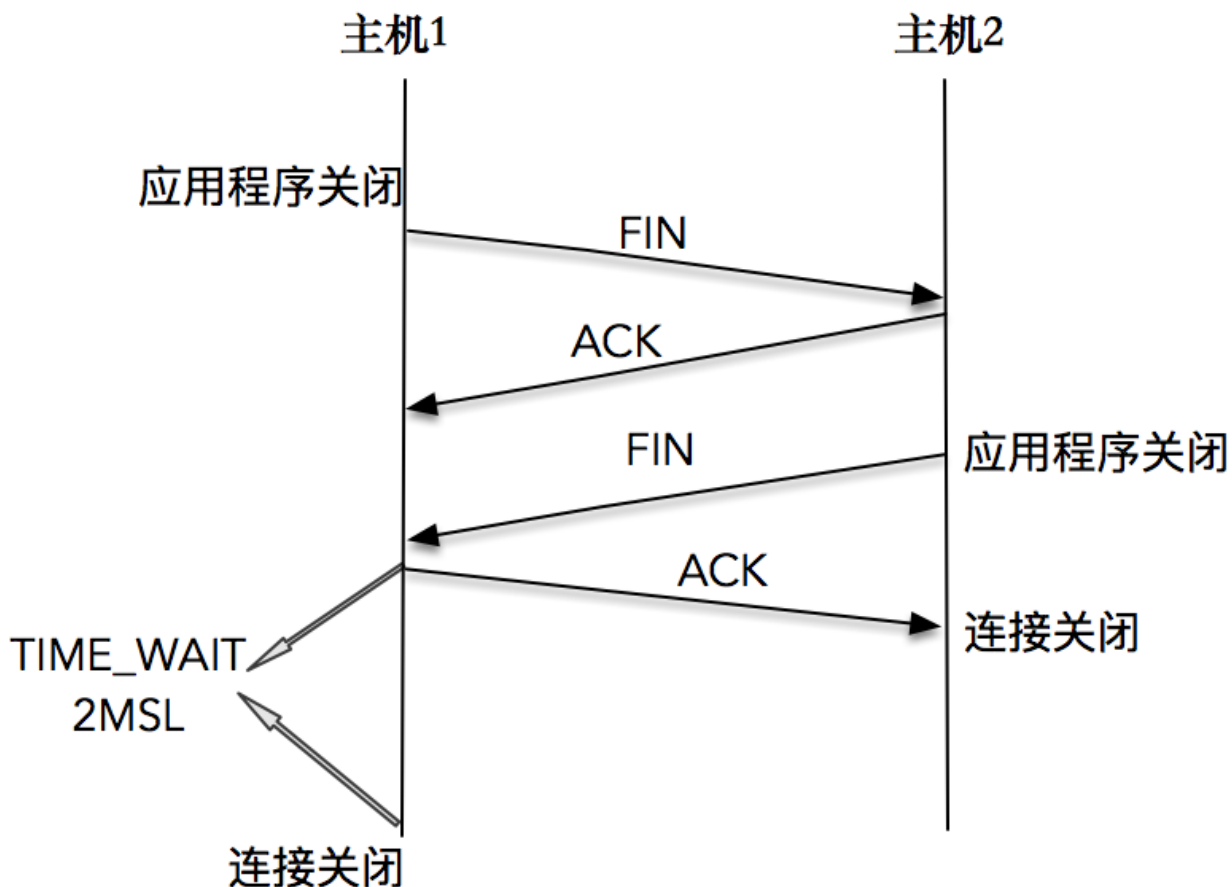
如果主机 1 没有维护 TIME\_WAIT 状态，而直接进入 CLOSED 状态，它就失去了当前状态的上下文，只能回复一个 RST 操作，从而导致被动关闭方出现错误。

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 protocol view
```

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 content 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，是怎么样的机制，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。




# 网络编程实战

从底层到实战，深度解析网络编程

盛延敏

前大众点评云平台首席架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。



上一篇 09 | 答疑篇：学习网络编程前，需要准备哪些东西？

下一篇 11 | 优雅地关闭还是粗暴地关闭？

## 精选留言 (18)

写留言



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？

1

3



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：)

展开

1

1



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

展开

1



永恒记忆

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tim...

展开

4

1



Berry Wang



2019-08-25

老师可以解释下 一个tcp连接对应一个端口的这句话吗？不太懂 正常我们启动一个web应用 启动结束端口就已经定好了 不同客户端发起访问访问到的都是同一个端口号

💬 1



沉淀的梦想

2019-08-24

使用SO\_REUSEADDR和SO\_REUSEPORT这两个选项是好的方法吗？

💬 1



Liam

2019-08-24

对于问题2，我觉得不需要统一时钟，因为timestamp的作用是给某端区分新旧连接数据，由端某生成并由该端进行判断，不会涉及双方的时钟问题。对time\_wait socket重用后，相当于创建了一条新的连接，这时候会设置一个新的timestamp，拿到数据宝后如果时间 < 新连接时间，则认为是旧连接，不处理即可。

展开 ▾



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...

展开 ▾

💬 2



Leon 📷

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



传说中的成大大

2019-08-23

关于第二个问题，思考了第二遍，因为是发送方的当前时间戳和从对方接受到的最新时间戳，如果接收方改了时间然后再从网络里面取出来自己上一次的发送的时间戳有很大差异

就会出问题了，比如往前改或者往后改！最关键我觉得还是看怎么通过时间戳判断生存时间！

展开 ∨



**传说中的成大大**

2019-08-23

第一问 是因为网络上的包在传输过程中过了最大生存时间则将会被丢弃，tcp/ip的一个处理机制

第二问 需要定义一个统一的时钟 不然如果发送双方的时间不一样的话 通过时间戳这个处理方式就会出问题

展开 ∨



**d**

2019-08-23

老师，如果time\_wait 默认是60s 的话，那2msl = time\_wait,那msl是否是30s? 还有化身和分节对应到英文中的什么单词，在这里不太清楚是什么意思



**在路上**

2019-08-23

我看到的答案设置方法都是setsockopt



**许童童**

2019-08-23

老师你好，TIME\_WAIT 是TCP主动关闭方才会存在的状态，什么机制确认谁是主动关闭方？



**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。





杨领well

2019-08-23

我觉得 MSL 就是发送方自以为是的认为，在发送过程中的路由器或主机是不知道 MSL 或者超时什么的。而它们是以 TTL 来作控制的。



老姜

2019-08-23

问题二。本来计算往返时间用的就是上一次自己的时间戳。。。。



在路上

2019-08-23

问题1：有个浅薄的想法，有个字段记录，比如我发送的时候，某个字段是0，正常接收到之后也应该为0，超时设置成其他的（没百度到具体怎么实现的）。

问题2：统一的时间应该还是很有必要的，或者保存好时间差不过感觉这个不大可行

展开 ∨

