**[理解TIME\_WAIT](http://www.firefoxbug.com/index.php/archives/2795/)**

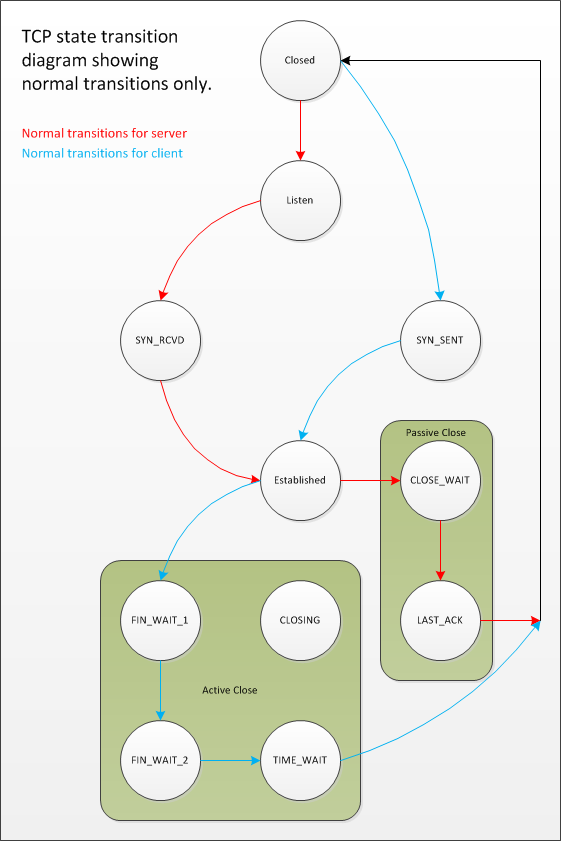
* 作者：firefoxbug

* 时间：November 6, 2014

* 分类：[TCP/IP](http://www.firefoxbug.com/index.php/category/tcpip/)

**前言**

TIME\_WAIT 是在TCP协议中很模糊的概念，它可能使socke能陷入的一种时间相对比较长的状态，过多的TIME\_WAIT会影响新socket的建立。TIME\_WAIT为什么会存在？它的作用又是什么？下面我们就来理解下TIME\_WAIT。



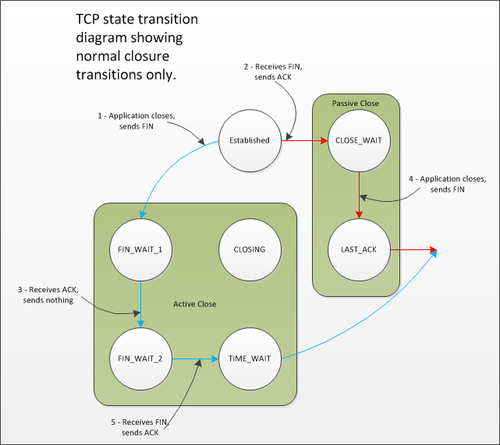
这张图详细的列出了TCP建立连接和断开连接的各个TCP状态之间的转换。红色的代表server，蓝色的代表client。下面列出各自的TCP状态转换条件

**TCP建立连接**

1. Client: 向server发送 SYN 包，表示请求建立连接，进入 SYN\_SENT 状态；
2. Server: 接收来自client的 SYN 包，发送 SYN/ACK 包，代表client->server单向tcp连接已经建立， 进入 SYN\_RCVD 状态；
3. Client: 接收到来自server的 SYN/ACK 包，发送给server ACK 包，进入 Established 状态；
4. Server: 收到client的 ACK 包，代表 server->client 的单向tcp连接也建立，此时进入 Established 状态；

**TCP断开连接**

先引入两个概念，首先调用close()是"主动关闭"(active close)，另一个是"被动关闭"(passive close)。一般我们连上ftp或者http，断开连接的都是客户端。看上面的图，"主动关闭"端状态要经历3个状态，而TIME\_WAIT是属于“主动关闭”端最后的一个tcp状态。



1. client: 主动调用close()，发送 FIN 包，此时client就"主动关闭"端，进入 FIN\_WAIT\_1 状态；
2. Server: server自然成为"被动关闭"端，收到来自client的 FIN 包，发送 ACK 包，代表client->server单向tcp连接已经关闭，进入 CLOSE\_WAIT 状态；
3. Client: 接收到来自server的 ACK 包，啥都不做，client->server单向的tcp连接已经断开，不能再发送应用层数据，进入 FIN\_WAIT\_2 状态；
4. Server: server端给client端发送 FIN 包，代表准备关闭server->client的tcp连接，server进入LAST\_ACK 状态；
5. Client: 收到来自server的 FIN 包，发送 ACK 包，此时进入 TIME\_WAIT 状态；
6. Server: 收到Client的 ACK 包，就进入closed状态，Server端此次socket tcp连接完全端口；
7. Client: 持续TIME\_WAIT状态"一段时间"；

**理解TIME\_WAIT**

理解了上面的原理之后，接着就是正式介绍TIME\_WAIT。TIME\_WAIT的时间大多数情况下都是2倍的MSL（Maximum Segment Lifetime），MSL是一个数据包在网络上能生存的最长生命周期，一旦超过MSL的包就会被丢弃。从上面可以看到，TIME\_WAIT是“主动关闭”端的最后一个状态，引入TIME\_WAIT的原因有:

1. 确保"主动关闭"端最后发出的 ACK 到达"被动关闭"端

2. 保证新tcp连接和老tcp连接不会干扰

**原因1: 确保"主动关闭"端最后发出的 ACK 到达"被动关闭"端**

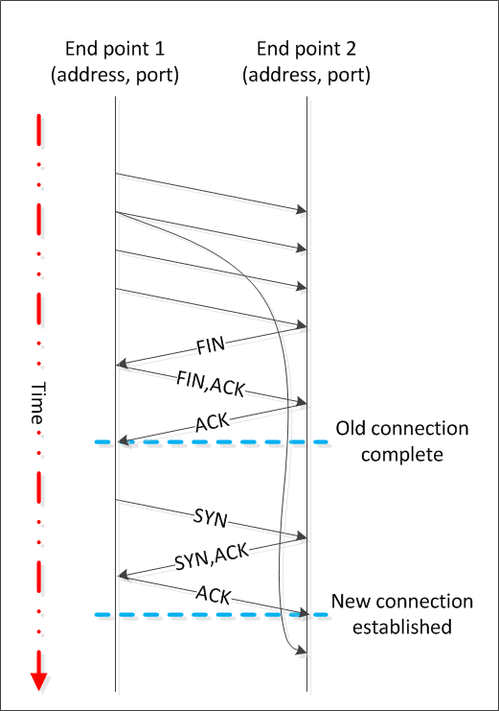
看上面tcp断开连接的图，由client主动调用close()，发出FIN包，然后接收到server的ACK/FIN包，客户端最后发一个FIN包，进入TIME\_WAIT。

设想一下，如果没有TIME\_WAIT，client端发送最后的FIN包后里面关闭连接，如果由于网络原因，最后发出的FIN包没有顺利到达server(此时的server一直处于LAST\_ACK状态等待最后FIN)，server长时间没有接收到FIN包，会认为之前由server发出的ACK/FIN包client没有收到，server会重新发送一个ACK/FIN包，这时候client收到ACK/FIN包，发现端口已经关闭，协议栈直接回复RST包，导致server端接收到RST包报错，影响应用进程。

所以 TIME\_WAIT 的作用可以保证最后的ACK包必然能到达对方，确保最后的连接正常端口。也解释了TIME\_WAIT时间是2\*MSL的原因。

**原因2: 保证新tcp连接和老tcp连接不会干扰**

看看下面的图



End Point2发送FIN包后，没有进入TIME\_WAIT状态，此时新的tcp请求又来了，而且src\_ip，src\_port，dst\_ip，dst\_port都是一样的，新的连接建立TCP请求后，老的连接包可能会干扰新连接的包，导致乱序。所以引入TIME\_WAIT，2\*MSL能让老连接的包彻底在网络中消失，保证新连接绝对干净。

**TIME\_WAIT数量多？**

TIME\_WAIT是占资源的，包括端口资源，协议栈队列，所以大量的TIME\_WAIT会影响socket建立新连接，这点特别在高性能的Web服务器中很讲究，那么有办法去减少TIME\_WAIT数量吗？嘿嘿，看了上面TIME\_WAIT存在的原因后，还想去调整tcp\_tw\_recycle或者tcp\_tw\_reuse等参数吗？这很有可能会引发未知的TCP错误，而且很诡异，很难排查。所以在高性能的Web服务器里面，必然会去设置HTTP的KeepAlive，不然Web服务器立马就被大量的TIME\_WAIT影响服务。

**调整TIME\_WAIT数量**

要调整TIME\_WAIT的数量，网上都是这几个参数，要修改的话还是悠着点吧？

net.ipv4.tcp\_tw\_reuse = 0 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；打开tcp\_tw\_reuse的时候要注意，是客户端还是服务端，如果是

net.ipv4.tcp\_tw\_recycle = 0 表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。

net.ipv4.tcp\_fin\_timeout = 30 表示如果套接字由本端要求关闭，这个参数决定了它保持在FIN-WAIT-2状态的时间。

**tcp\_tw\_recycle**

tcp\_tw\_recycle必须和tcp\_timestamps一起打开，默认情况linux的tcp\_timestamps都是打开的，tcp\_tw\_recycle到底是多久回收sockets？正常是700ms。

tcp\_tw\_recycle的坑：当多个客户端通过NAT方式联网并与服务端交互时，服务端看到的是同一个IP，也就是说对服务端而言这些客户端实际上等同于一个，可惜由于这些客户端的时间戳可能存在差异，于是乎从服务端的视角看，便可能出现时间戳错乱的现象，进而直接导致时间戳小的数据包被丢弃

注意：在NAT模型中，tcp\_tw\_recycle打开可能会导致丢包

**tcp\_tw\_reuse**

1. tcp\_tw\_reuse选项和tcp\_timestamps选项也必须同时打开；
2. 重用TIME\_WAIT的条件是收到最后一个包后超过1s

**总结**

TIME\_WATI出现在TCP连接"主动关闭"端，理论上会持续2\*MSL(根据不通系统而定，因为IP有TTL)，TIME\_WAIT的出现是为了解决两个问题

1. 确保连接能正确断开(确保"主动关闭"端最后发出的 ACK 到达"被动关闭"端)

2. 确保新的tcp连接和老的tcp连接不会干扰