**再叙TIME\_WAIT**

发表于[**2013-12-31**](https://huoding.com/2013/12/31/316)

之所以起这样一个题目是因为很久以前我曾经写过一篇介绍TIME\_WAIT的[文章](https://huoding.com/2012/01/19/142)，不过当时基本属于浅尝辄止，并没深入说明问题的来龙去脉，碰巧这段时间反复被别人问到相关的问题，让我觉得有必要全面总结一下，以备不时之需。

讨论前大家可以拿手头的服务器摸摸底，记住「ss」比「netstat」快：

shell> ss -ant | awk '

NR>1 {++s[$1]} END {for(k in s) print k,s[k]}

'

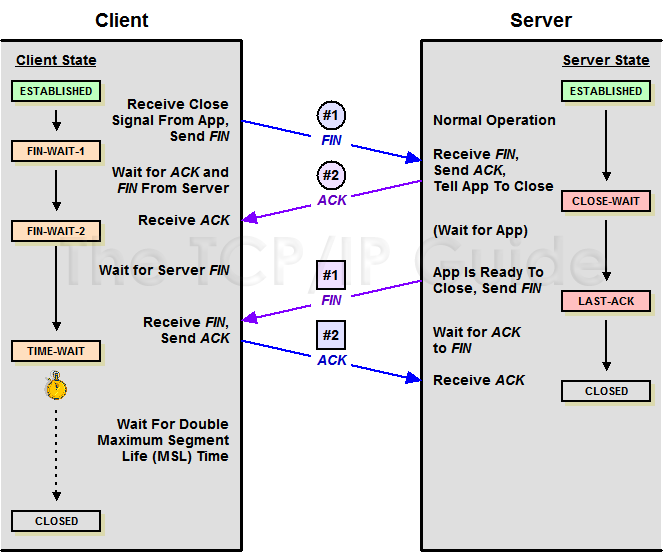
如果你只是想单独查询一下TIME\_WAIT的数量，那么还可以更简单一些：

shell> cat /proc/net/sockstat

我猜你一定被巨大无比的TIME\_WAIT网络连接总数吓到了！以我个人的经验，对于一台繁忙的Web服务器来说，如果主要以短连接为主，那么其TIME\_WAIT网络连接总数很可能会达到几万，甚至十几万。虽然一个TIME\_WAIT网络连接耗费的资源无非就是一个端口、一点内存，但是架不住基数大，所以这始终是一个需要面对的问题。

**为什么会存在TIME\_WAIT？**

TCP在建立连接的时候需要握手，同理，在关闭连接的时候也需要握手。为了更直观的说明关闭连接时握手的过程，我们引用「[The TCP/IP Guide](http://www.tcpipguide.com/free/index.htm)」中的[例子](http://www.tcpipguide.com/free/t_TCPConnectionTermination.htm)：

[](https://huoding.com/wp-content/uploads/2013/12/tcp_close.png)

TCP Close

因为TCP连接是双向的，所以在关闭连接的时候，两个方向各自都需要关闭。先发FIN包的一方执行的是主动关闭；后发FIN包的一方执行的是被动关闭。主动关闭的一方会进入TIME\_WAIT状态，并且在此状态停留两倍的[MSL](http://en.wikipedia.org/wiki/Maximum_Segment_Lifetime)时长。

穿插一点MSL的知识：MSL指的是报文段的最大生存时间，如果报文段在网络活动了MSL时间，还没有被接收，那么会被丢弃。关于MSL的大小，[RFC 793](http://tools.ietf.org/search/rfc793)协议中给出的建议是两分钟，不过实际上不同的操作系统可能有不同的设置，以Linux为例，通常是半分钟，两倍的MSL就是一分钟，也就是60秒，并且这个数值是硬编码在[内核](https://github.com/torvalds/linux/blob/master/include/net/tcp.h)中的，也就是说除非你重新编译内核，否则没法修改它：

#define TCP\_TIMEWAIT\_LEN (60\*HZ)

如果每秒的连接数是一千的话，那么一分钟就可能会产生六万个TIME\_WAIT。

为什么主动关闭的一方不直接进入CLOSED状态，而是进入TIME\_WAIT状态，并且停留两倍的MSL时长呢？这是因为TCP是建立在不可靠网络上的可靠的协议。例子：主动关闭的一方收到被动关闭的一方发出的FIN包后，回应ACK包，同时进入TIME\_WAIT状态，但是因为网络原因，主动关闭的一方发送的这个ACK包很可能延迟，从而触发被动连接一方重传FIN包。极端情况下，这一去一回，就是两倍的MSL时长。如果主动关闭的一方跳过TIME\_WAIT直接进入CLOSED，或者在TIME\_WAIT停留的时长不足两倍的MSL，那么当被动关闭的一方早先发出的延迟包到达后，就可能出现类似下面的问题：

* 旧的TCP连接已经不存在了，系统此时只能返回RST包
* 新的TCP连接被建立起来了，延迟包可能干扰新的连接

不管是哪种情况都会让TCP不再可靠，所以TIME\_WAIT状态有存在的必要性。

**如何控制TIME\_WAIT的数量？**

从前面的描述我们可以得出这样的结论：TIME\_WAIT这东西没有的话不行，不过太多可能也是个麻烦事。下面让我们看看有哪些方法可以控制TIME\_WAIT数量，这里只说一些常规方法，另外一些诸如SO\_LINGER之类的方法太过偏门，略过不谈。

**ip\_conntrack**：顾名思义就是跟踪连接。一旦激活了此模块，就能在系统参数里发现很多用来控制网络连接状态超时的设置，其中自然也包括TIME\_WAIT：

shell> modprobe ip\_conntrack

shell> sysctl net.ipv4.netfilter.ip\_conntrack\_tcp\_timeout\_time\_wait

我们可以尝试缩小它的设置，比如十秒，甚至一秒，具体设置成多少合适取决于网络情况而定，当然也可以参考相关的[案例](http://blog.engineyard.com/2012/linux-scalability)。不过就我的个人意见来说，ip\_conntrack引入的问题比解决的还多，比如性能会大幅下降，所以不建议使用。

**tcp\_tw\_recycle**：顾名思义就是回收TIME\_WAIT连接。可以说这个内核参数已经变成了大众处理TIME\_WAIT的万金油，如果你在网络上搜索TIME\_WAIT的解决方案，十有八九会推荐设置它，不过这里隐藏着一个不易察觉的[陷阱](http://www.litrin.net/2013/03/01/android%E4%B9%8B%E7%BD%91%E7%BB%9C%E4%B8%A2%E5%8C%85%E4%BA%8B%E4%BB%B6/)：

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。参考：[tcp\_tw\_recycle和tcp\_timestamps导致connect失败问题](http://blog.sina.com.cn/s/blog_781b0c850100znjd.html)。

**tcp\_tw\_reuse**：顾名思义就是复用TIME\_WAIT连接。当创建新连接的时候，如果可能的话会考虑复用相应的TIME\_WAIT连接。通常认为「tcp\_tw\_reuse」比「tcp\_tw\_recycle」安全一些，这是因为一来TIME\_WAIT创建时间必须超过一秒才可能会被复用；二来只有连接的时间戳是递增的时候才会被复用。[官方文档](https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt)里是这样说的：如果从协议视角看它是安全的，那么就可以使用。这简直就是外交辞令啊！按我的看法，如果网络比较稳定，比如都是内网连接，那么就可以尝试使用。

不过需要注意的是在哪里使用，既然我们要复用连接，那么当然应该在连接的发起方使用，而不能在被连接方使用。举例来说：客户端向服务端发起HTTP请求，服务端响应后主动关闭连接，于是TIME\_WAIT便留在了服务端，此类情况使用「tcp\_tw\_reuse」是无效的，因为服务端是被连接方，所以不存在复用连接一说。让我们延伸一点来看，比如说服务端是PHP，它查询另一个MySQL服务端，然后主动断开连接，于是TIME\_WAIT就落在了PHP一侧，此类情况下使用「tcp\_tw\_reuse」是有效的，因为此时PHP相对于MySQL而言是客户端，它是连接的发起方，所以可以复用连接。

说明：如果使用tcp\_tw\_reuse，请激活tcp\_timestamps，否则无效。

**tcp\_max\_tw\_buckets**：顾名思义就是控制TIME\_WAIT总数。[官网文档](https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt)说这个选项只是为了阻止一些简单的DoS攻击，平常不要人为的降低它。如果缩小了它，那么系统会将多余的TIME\_WAIT删除掉，日志里会显示：「TCP: time wait bucket table overflow」。

需要提醒大家的是物极必反，曾经看到有人把「tcp\_max\_tw\_buckets」设置成0，也就是说完全抛弃TIME\_WAIT，这就有些冒险了，用一句围棋谚语来说：入界宜缓。

…

有时候，如果我们换个角度去看问题，往往能得到四两拨千斤的效果。前面提到的例子：客户端向服务端发起HTTP请求，服务端响应后主动关闭连接，于是TIME\_WAIT便留在了服务端。这里的关键在于主动关闭连接的是服务端！在关闭TCP连接的时候，先出手的一方注定逃不开TIME\_WAIT的宿命，套用一句歌词：把我的悲伤留给自己，你的美丽让你带走。如果客户端可控的话，那么在服务端打开[KeepAlive](http://en.wikipedia.org/wiki/HTTP_persistent_connection)，尽可能不让服务端主动关闭连接，而让客户端主动关闭连接，如此一来问题便迎刃而解了。