TCP协议知识点

目录

[TCP协议知识点 1](#_Toc532286011)

[1.FIN\_Wait1 2](#_Toc532286012)

[1.1.FIN\_WAIT1是啥？ 2](#_Toc532286013)

[1.2.FIN\_WAIT1是怎么来的： 2](#_Toc532286014)

[1.3.FIN\_WAIT1持续时间： 2](#_Toc532286015)

[1.4.FIN\_WAIT1不正常关闭引出的问题： 3](#_Toc532286016)

[1.5.FIN\_WAIT1的控制点 3](#_Toc532286017)

[1.5.1.tcp\_orphan\_retries 3](#_Toc532286018)

[1.5.2.tcp\_fin\_timeout 4](#_Toc532286019)

[2. TIME\_WAIT 5](#_Toc532286020)

[为什么要用TIME\_WAIT状态： 5](#_Toc532286021)

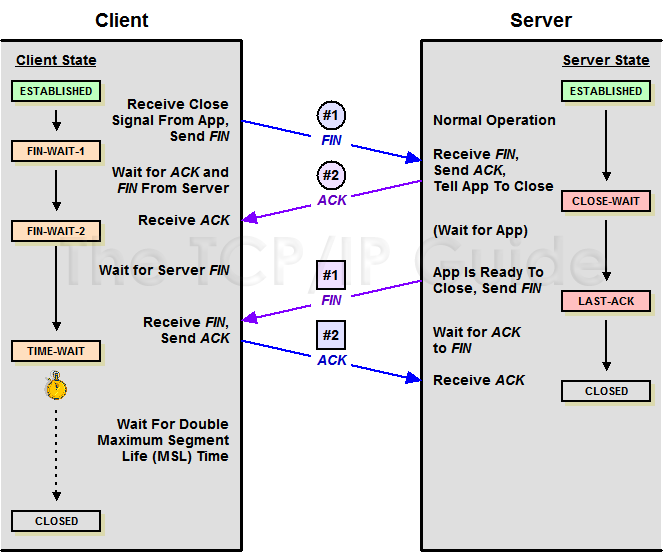
[3相关命令 6](#_Toc532286022)

# 1.FIN\_Wait1

## 1.1.FIN\_WAIT1是啥？

FIN\_WAIT1是发生在TCP关闭连接时的一种状态。发生在主动关闭方的第一个动作之后。

## 1.2.FIN\_WAIT1是怎么来的：



看图可知FIN\_WAIT1产生的过程：

1：**主动关闭的一方发出 FIN**，**同时进入 FIN\_WAIT1 状态**；

2：被动关闭的一方响应 ACK，从而使主动关闭的一方迁移至 FIN\_WAIT2 状态；

3：接着被动关闭的一方同样会发出 FIN，主动关闭的一方响应 ACK，同时迁移至 **TIME**\_WAIT 状态。

## 1.3.FIN\_WAIT1持续时间：

FIN\_WAIT1 能持续多久？ 一般情况下，服务器间的 ACK 确认是非常快的，以至于凭肉眼往往观察不到 FIN\_WAIT1 的存在。FIN\_WAIT1的存活时间取决于通讯的对端回复ACK的时间，如果对端已经关闭或者窗口大小不对，不回复ACK则会导致FIN\_WAIT1长时间持续。

## 1.4.FIN\_WAIT1不正常关闭引出的问题：

如上所言：服务器发送FIN试图关闭连接时对端不响应ACK，服务器就进入FIN\_WAIT1状态，产生一个孤儿连接orphaned。如果大多ACK不回复，则会导致服务器FIN\_WAIT1增多。大量的FIN\_WAIT1会消耗服务器资源，造成DDOS攻击。

[root@ip-172-31-10-242 all\_log]# ss -s

Total: 2445 (kernel 2752)

TCP: 2309 (estab 2167, closed 92, orphaned 574, synrecv 0, timewait 59/0), ports 0

Transport Total IP IPv6

\* 2752 - -

RAW 0 0 0

UDP 6 4 2

TCP 2217 54 2163

INET 2223 58 2165

FRAG 0 0 0

## 1.5.FIN\_WAIT1的控制点

### 1.5.1.tcp\_orphan\_retries

计算机是无法处理无限，无穷这种抽线的数学概念的，所有如果针对FIN的ACK迟迟不来，那么必然要有一个等待的极限，这个极限在Linux内核协议栈中由以下参数控制：

net.ipv4.tcp\_orphan\_retries # 默认值是0！这里有坑...

这个参数表示如果一直都收不到针对FIN的ACK，那么在彻底销毁这个FIN\_WAIT1的连接前，等待几轮RTO退避。

所谓的orphan tcp connection，意思就是说，在Linux进程层面，创建该连接的进程已经退出销毁了，然而在TCP协议层面，它依然在遵循TCP状态机的转换规则存在着。

注意，这个参数不是一个时间量，而是一个次数量。我们知道，TCP每一次超时，都会对下一次超时时间进行指数退避，这里的次数量就是要经过几次退避的时间。举一个例子，如果RTO是2ms，而tcp\_orphan\_retries 的值是4，那么所计算出的FIN\_WAIT1容忍时间就是： T=21+22+23+24

还是看看Linux内核文档怎么说的吧：

tcp\_orphan\_retries - INTEGER

  This value influences the timeout of a locally closed TCP connection, when RTO retransmissions remain unacknowledged.

  See tcp\_retries2 for more details.

  The default value is 8.

  If your machine is a loaded WEB server,

  you should think about lowering this value, such sockets

  may consume significant resources. Cf. tcp\_max\_orphans.

让我们看看tcp\_retries2，以获取数值的含义：

tcp\_retries2 - INTEGER

  This value influences the timeout of an alive TCP connection,

  when RTO retransmissions remain unacknowledged.

  Given a value of N, a hypothetical TCP connection following

  exponential backoff with an initial RTO of TCP\_RTO\_MIN would

  retransmit N times before killing the connection at the (N+1)th RTO.

  The default value of 15 yields a hypothetical timeout of 924.6

  seconds and is a lower bound for the effective timeout.

  TCP will effectively time out at the first RTO which exceeds the hypothetical timeout.

  RFC 1122 recommends at least 100 seconds for the timeout,

  which corresponds to a value of at least 8.

虽然说文档上默认值的建议是8，但是大多数的Linux发行版上其默认值都是0。更多详情，就自己看RFC和Linux源码吧。

有了这个参数保底，我们知道，即便是ACK永远不来，FIN\_WAIT1状态也不会一直持续下去的，这有效避免了有针对性截获ACK或者不发送ACK而导致的DDoS，退一万步讲，即便是没有DDoS，这种做法也具有资源利用率的容错性，使得资源使用更加高效。

如果主动断开端调用了close关掉了进程，它会进入FIN\_WAIT1状态，此时如果它再也收不到ACK，无论是针对pending在发送缓冲的数据还是FIN，它都会尝试重新发送，在收到ACK前会尝试N次退避，该N由tcp\_orphan\_retries参数控制。

从实际运维角度来看，需要注意以下参数设置：

net.ipv4.tcp\_synack\_retries = 4

net.ipv4.tcp\_syn\_retries = 3

net.ipv4.tcp\_orphan\_retries = 3

vi /etc/sysctl.conf

### 1.5.2.tcp\_fin\_timeout

最常见的误解是认为 tcp\_fin\_timeout 控制 FIN\_WAIT1 的过期，从名字上看也很像，但实际上它控制的是FIN\_WAIT2 的过期时间，官方文档是这样说的：

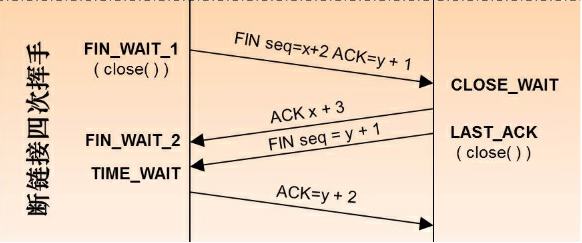
The length of time an orphaned (no longer referenced by any application)connection will remain in the FIN\_WAIT\_2 state before it is aborted at the local end. While a perfectly valid “receive only” state for an un-orphaned connection, an orphaned connection in FIN\_WAIT\_2 state could otherwise wait forever for the remote to close its end of the connection.

Cf. tcp\_max\_orphans

Default: 60 seconds

## 2. TIME\_WAIT

从上节图中可以看出，在关闭TCP时，状态切换流程为：FIN\_WAIT1-->FIN\_WAIT2-->TIME\_WAIT。TIME\_WAIT发生在最后关闭TCP的最后一个环节。



### 为什么要用TIME\_WAIT状态：

先考虑这样的一个情况，假如最后回应的ACK丢失了，也就是服务器端接收不到这个ACK，那么服务器将继续发送它最终的那个FIN，因此客户端必须维护状态信息（TIME\_WAIT）允许它重发最后的那个ACK。如果没有这个TIME\_WAIT状态，客户端处于CLOSED状态（开头就说了CLOSED状态实际并不存在，是我们为了方便描述假想的），那么客户端将响应RST，服务器端收到后会将该RST分节解释成一个错误，也就不能实现最后的全双工关闭了（可能是主动方单方的关闭）。所以要实现TCP全双工连接的正常终止（两方都关闭连接），必须处理终止过程中四个分节任何一个分节的丢失情况，那么主动关闭连接的主动端必须维持TIME\_WAIT状态，最后一个回应ACK的是主动执行关闭的那端。从变迁图可以看出，如果没有TIME\_WAIT状态，我们将没有任何机制来保证最后一个ACK能够正常到达。前面的FIN，ACK正常到达均有相应的状态对应。

还有这样一种情况，如果目前的通信双方都已经调用了 close()，都到达了CLOSED状态，没有TIME\_WAIT状态时，会出现这样一种情况，现在有一个新的连接被建立起来，使用的IP地址和端口和这个先前到达了CLOSED状态的完全相同，假定原先的连接中还有数据报文残存在网络之中，这样新的连接建立以后传输的数据极有可能就是原先的连接的数据报文，为了防止这一点，**TCP不允许从处于TIME\_WAIT状态的socket 建立一个连接**。处于TIME\_WAIT状态的 socket 在等待了两倍的MSL时间之后，将会转变为CLOSED状态。这里TIME\_WAIT状态持续的时间是2MSL（MSL是任何IP数据报能够在因特网中存活的最长时间），足以让这两个方向上的数据包被丢弃（最长是2MSL）。通过实施这个规则，我们就能保证每成功建立一个TCP连接时，来自该连接先前化身的老的重复分组都已经在网络中消逝了。

 综上来看：TIME\_WAIT存在的两个理由就是：

* 可靠地实现TCP全双工连接的终止；
* 允许老的重复分节（数据报）在网络中消逝。

### TIME\_WAIT的控制点

## 3相关命令

除了上面提到的ss -s 外还有以下命令：

* 分析哪种tcp状态数量异常

[root@ip-172-31-10-242 all\_log]# netstat -nat|awk '{print awk $NF}'|sort|uniq -c|sort -n

1 CLOSE\_WAIT

1 established)

1 State

1 SYN\_RECV

27 LISTEN

30 TIME\_WAIT

574 FIN\_WAIT1

2124 ESTABLISHED

[root@ip-172-31-10-242 all\_log]# netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

LAST\_ACK 4

SYN\_RECV 3

ESTABLISHED 2147

FIN\_WAIT1 574

TIME\_WAIT 30

* 请求8101服务的client ip按照连接数排序：

netstat -nat|grep ":8101"|awk '{print $5}' |awk -F: '{print $1}' | sort| uniq -c|sort -n

* w命令查看CPU占用，Load Average 显示一段时间 (1 分钟、5分钟、15分钟) 内平均 Load

[deve@r730-104 tomcat]$ w

06:31:37 up 12 days, 22:46, 4 users, load average: 48.06, 48.09, 48.12

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

root tty1 27Nov18 12days 0.30s 0.30s -bash

deve pts/0 218.17.179.250 30Nov18 1.00s 29:55m 0.29s sshd: deve [priv]

root pts/1 218.17.179.250 03Dec18 9.00s 270days 0.01s -bash

deve pts/2 218.17.179.250 28Nov18 4days 0.13s 0.13s -bash

