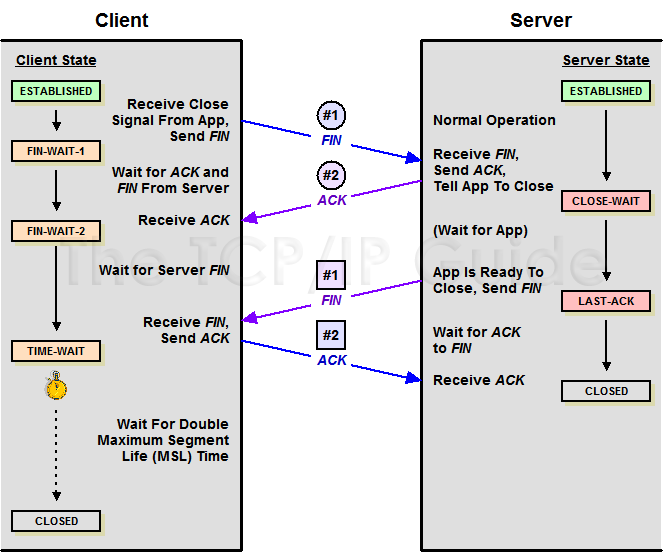
### **CLOSE\_WAIT**

TCP关闭连接时四次挥手的过程,如下图所示(图来自网络):  


有图可知,主动方发起关闭请求也就是FIN包后,被动方接收到包,**被动方接着进入**CLOSE\_WAIT**状态**,接着被动方发送FIN包告知主动方自己已关闭后进入LAST\_ACK状态.  
那么当被动方这个FIN包没有发送成功,那么其就一直处于CLOSE\_WAIT状态.那么问题成功转换为以下几个小问题:

**大量**CLOSE\_WAIT**有什么危害?**  
CLOSE\_WAIT状态不会自己消失,除非对应的应用进程死掉,不会消失就意味着一直占用服务器资源,端口总数又只有65535,因此这里的服务器作为连接的发起者就会造成大量端口被占用,一旦占用完就导致后面的请求都发不出去,也就是一开始图上另一个项目发请求出现的Address already in use (Bind failed)错误.

**被动方什么情况下**FIN**包会发送失败?**

* + 程序问题：如果代码层面忘记了 close 相应的 socket 连接，那么自然不会发出 FIN 包，从而导致 CLOSE\_WAIT 累积；或者代码不严谨，出现死循环之类的问题，导致即便后面写了 close 也永远执行不到。
  + 响应太慢或者超时设置过小：如果连接双方不和谐，一方不耐烦直接 timeout，另一方却还在忙于耗时逻辑，就会导致 close 被延后。响应太慢是首要问题，不过换个角度看，也可能是 timeout 设置过小。
  + BACKLOG 太大：此处的 backlog 不是 syn backlog，而是 accept 的 backlog，如果 backlog 太大的话，设想突然遭遇大访问量的话，即便响应速度不慢，也可能出现来不及消费的情况，导致多余的请求还在队列里就被对方关闭了。

### **解决问题**

知道了产生的原因,自然好解决,根据netstat给出的信息包括pid定位到具体的应用,然后通过git查看最近代码改动,最终找到之前上线的一段代码使用了python的httplib,使用完却**没有主动close释放连接**,因此出现了这个问题.

那么**为什么HttpClient访问时端口会分配到CLOSE\_WAIT对应的端口?**  
Linux会为每一次请求分配临时端口,这个分配范围在/proc/sys/net/ipv4/ip\_local\_port\_range中有记录,在我这台服务器上其值是20000-65535,大量的CLOSE\_WAIT就会导致可分配的端口数减少,因此系统会在指定范围内选择一个没有冲突的端口,一旦端口消耗完毕就会造成冲突.也就是上面的错误Address already in use (Bind failed).

### **TIME\_WAIT**

上面结果图中TIME\_WAIT也有几百个,这个是什么原因?  
对于四次挥手过程中,当主动方接收到被动放的关闭确认信号FIN后,主动方会回复一个ACK信号,然后会进入TIME\_WAIT状态,此时会等待2MLS,在Linux中也就是60s,因此相对上述2000多个活跃tcp来说,这100多的TIME\_WAIT是正常现象.  
然后为什么TCP主动方关闭后需要等待2MLS?  
因为TCP是可靠的通信,在主动方回复ACK时如果由于网络问题该包发送失败,那么被动方就会进行FIN重传,此时重传会遇到两个场景:

* 主动方已关闭,旧的TCP连接已经消失,那么系统只能回复RST包.
* 主动方已关闭,然后利用此端口建立了新的连接.也就是旧的TCP关闭,新的TCP已建立,那么就会造成信道的不可靠.

因此超时等待机制是必要的