

**计算机网络分析报告**

**学生姓名**  付炎平

**学 号**  2019217819

**专业班级**  物联网工程19-2班

**指导教师**  周健

**合肥工业大学 计算机与信息学院**

**关于不正常的连接释放请求**

**1、当通信双方同时发出连接释放请求**

由于双方都发送了FIN=1，两方的连接已经释放,不管通信双方是否有数据传送给对方,这些数据都会丢失，就没有四次挥手中数据传送的阶段，通信双方也会对对方的连接释放请求进行确认。

**2、A/B建立连接，B又收到A迟到的连接释放请求**

A端的拒绝是因为之前发起的连接释放请求由于超时而被重置，故而认为没有发起连接释放请求，而B端的拒绝是因为A端没有对B端同意释放连接进行确认故放弃连接释放，因此，这种情况不会使A和B的连接进行释放。

**3、突然连接释放，其中一个用户关闭数据传输的两个方向**

突然连接释放机制非常简单，依赖于设置了RST位的单个段。可以出于以下原因发送包含RST位的TCP 段：

①为不存在的 TCP 连接接收到非SYN段[RFC 793](http://tools.ietf.org/html/rfc793.html" \t "https://www.opentextbooks.org.hk/ditatopic/_blank)。

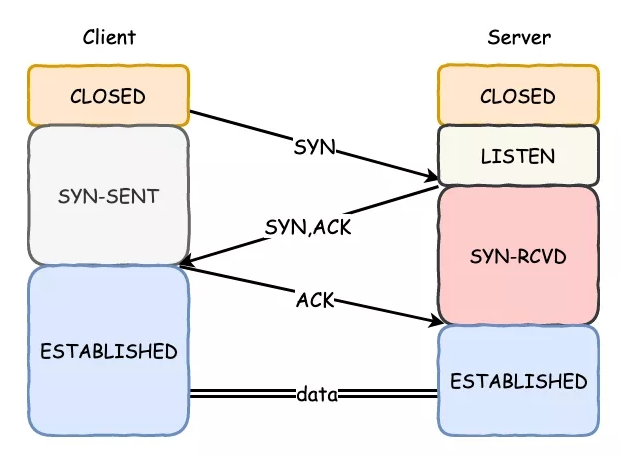
②通过扩展，一些实现使用 RST 段响应在现有连接上接收但具有无效标头[RFC 3360 的段](http://tools.ietf.org/html/rfc3360.html" \t "https://www.opentextbooks.org.hk/ditatopic/_blank)。这会导致相应的连接被关闭并导致安全攻击[RFC 4953](http://tools.ietf.org/html/rfc4953" \t "https://www.opentextbooks.org.hk/ditatopic/_blank) 扩展，一些实现在需要关闭现有 TCP 连接时发送 RST 段（例如，因为没有足够的资源来支持此连接或因为远程主机被认为是不可达的）。测量表明，这种 TCP RST 的使用很普遍。

当一个 RST 段由 TCP 实体发送时，它应该包含连接的序列号的当前值（如果它不属于任何现有连接，则为 0）并且确认号应该设置为下一个预期的输入此连接上的序列号。

**4、客户端意外断开**

双方并未按照协议上的四次挥手去断开连接，一般的处理办法都是利用保活机制。在A和B建立连接后，若双方均不发送数据只保持连接，则再两小时后系统会自动启动保活机制向peer发送包，看对方是否回应ack，若可以收到则继续保持，否则无效。

**5、挥手异常**

首先是经典的TCP四次挥手的消息和双端状态的变化。

**(1) 断开连接的 FIN 包丢了。**

如果一个包发出去，在一定时间内，只要没有收到对端的「ACK」回复，均认为这个包丢了，会触发超时重传机制。而不会关心到底是自己发的包丢了，还是对方的「ACK」丢了。所以在这里，如果客户端率先发的「FIN」包丢了，或者没有收到对端的「ACK」回复，则会触发超时重传，直到触发重传的次数，直接关闭连接。对于服务端而言，如果客户端发来的「FIN」没有收到，就没有任何感知。会在一段时间后，也关闭连接。

**(2) 服务端第一次回复的 ACK 丢了。**

此时因为客户端没有收到「ACK」应答，会尝试重传之前的「FIN」请求，服务端收到后，又会再重传「ACK」。而此时服务端已经进入 CLOSED-WAIT 状态，开始做断开连接前的准备工作。当准备好之后，会回复「FIN,ACK」，注意这个消息是携带了之前「ACK」的响应序号的。只要这个消息没丢，客户端可以凭借「FIN,ACK」包中的响应序号，直接从 FIN-WAIT-1 状态，进入 TIME-WAIT 状态，开始长达 2MSL 的等待。

**(3) 服务端发送的 FIN,ACK 丢了。**

服务端在超时后会重传，此时客户端有两种情况，要么处于 FIN-WAIT-2 状态(之前的 ACK 也丢了)，会一直等待;要么处于 TIME-WAIT 状态，会等待 2MSL 时间。也就是说，在之后的一小段时间内客户端还在，客户端在收到服务端发来的「FIN,ACK」包后，也会回复一个「ACK」应答，并做自己的状态切换。

**(4) 客户端最后回复的 ACK 丢了。**

客户端在回复「ACK」后，会进入 TIME-WAIT 状态，并开始长达 2MSL 的等待，服务端因为没有收到「ACK」的回复，会重试一段时间，直到服务端重试超时后主动断开。或者等待新的客户端接入后，收到服务端重试的「FIN」消息后，回复「RST」消息，在收到「RST」消息后，复位服务端的状态。

**(5) 客户端收到 ACK 后，服务端跑路了。**

客户端在收到「ACK」后，进入了 FIN-WAIT-2 状态，等待服务端发来的「FIN」包，而如果服务端跑路了，这个包永远都等不到。在 TCP 协议中，是没有对这个状态的处理机制的。但是协议不管，系统来凑，操作系统会接管这个状态，例如在 Linux 下，就可以通过 tcp\_fin\_timeout 参数，来对这个状态设定一个超时时间。

**(6) 客户端收到 ACK 后，客户端自己跑路了。**

客户端收到「ACK」后直接跑路，服务端后续在发送的「FIN,ACK」就没有接收端，也就不会得到回复，会不断的走 TCP 的超时重试的机制，此时服务端处于 LAST-ACK 状态。那就要分 2 种情况分析：在超过一定时间后，服务端主动断开；收到「RST」后，主动断开连接。

「RST」消息是一种重置消息，表示当前错误了，应该回到初始的状态。如果客户端跑路后有新的客户端接入，会在此发送「SYN」以期望建立连接，此时这个「SYN」将被忽略，并直接回复「FIN,ACK」消息，新客户端在收到「FIN」消息后是不会认的，并且会回复一个「RST」消息。