不为人知的网络编程(四)：深入研究分析TCP的异常关闭-网络编程/专项技术区 - 即时通讯开发者社区!

原作者：腾讯互娱谢代斌，由即时通讯网重新整理发布，感谢原作者的无私分享。

1、前言

研究测试TCP断开和异常的各种情况，以便于分析网络应用（比如tconnd）断网的原因和场景，帮组分析和定位连接异常掉线的问题，并提供给TCP相关的开发测试人员作为参考。

各个游戏接入都存在一定的掉线问题，而且有的游戏项目的掉线比例还比较高，现在腾讯互娱自研游戏的网络接入基本上都用的是tconnd和ProtocalHandler组件（该组件请参考附件《TSF4G_ProtocalHandler开发指导手册》），

因此有幸参与了其掉线原因分析和研究。

在参与A项目的掉线问题研究分析过程中，tconnd增加了玩家每个连接的流水日志和ProtocalHandler增加了每个连接的Qos上报日志，通过这些日志记录了每一次连接的断开原因和相关统计数据，其中包括了连接异常断开时TCP的底层错误码。

通过对tconnd的流水日志和ProtocalHandler的Qos日志进行统计分析，发现连接异常断开时TCP的错误码大部分是“104: Connection reset by peer”（Linux下）或“10054: An existing connection was forcibly closed by the remote host”（Windows下），单纯从错误码本来来说，大家都明白是“网络被对端重置了”，但究竟什么情况下会导致这种情况呢？因此本文就对TCP的各种关闭情况做了进一步的测试研究。

2、系列文章

本文是系列文章中的第4篇，本系列文章的大纲如下：

- 《不为人知的网络编程(一)：浅析TCP协议中的疑难杂症(上篇)》
- 《不为人知的网络编程(二)：浅析TCP协议中的疑难杂症(下篇)》
- 《不为人知的网络编程(三)：关闭TCP连接时为什么会TIME_WAIT、CLOSE_WAIT》
- 《不为人知的网络编程(四)：深入研究分析TCP的异

常关闭》（本文）

- 《不为人知的网络编程(五)：UDP的连接性和负载均衡》
- 《不为人知的网络编程(六)：深入地理解UDP协议并用好它》
- 《不为人知的网络编程(七)：如何让不可靠的UDP变的可靠？》
- 《不为人知的网络编程(八)：从数据传输层深度解密HTTP》
- 《不为人知的网络编程(九)：理论联系实际，全方位深入理解DNS》

如果您觉得本系列文章过于专业，您可先阅读《网络编程懒人入门》系列文章，该系列目录如下：

- 《网络编程懒人入门(一)：快速理解网络通信协议（上篇）》
- 《网络编程懒人入门(二)：快速理解网络通信协议（下篇）》
- 《网络编程懒人入门(三)：快速理解TCP协议一篇就够》
- 《网络编程懒人入门(四)：快速理解TCP和UDP的差异》
- 《网络编程懒人入门(五)：快速理解为什么说UDP有时比TCP更有优势》

本站的《脑残式网络编程入门》也适合入门学习，本系列

大纲如下：

- 《脑残式网络编程入门(一)：跟着动画来学TCP三次握手和四次挥手》
- 《脑残式网络编程入门(二)：我们在读写Socket时，究竟在读写什么？》
- 《脑残式网络编程入门(三)：HTTP协议必知必会的一些知识》
- 《脑残式网络编程入门(四)：快速理解HTTP/2的服务器推送(Server Push)》

关于移动端网络特性及优化手段的总结性文章请见：

- 《现代移动端网络短连接的优化手段总结：请求速度、弱网适应、安全保障》
- 《移动端IM开发者必读(一)：通俗易懂，理解移动网络的“弱”和“慢”》
- 《移动端IM开发者必读(二)：史上最全移动弱网络优化方法总结》

3、参考资料

《TCP/IP详解 - 第11章·UDP：用户数据报协议》
《TCP/IP详解 - 第17章·TCP：传输控制协议》
《TCP/IP详解 - 第18章·TCP连接的建立与终止》

《TCP/IP详解 - 第21章·TCP的超时与重传》
《通俗易懂-深入理解TCP协议（上）：理论基础》
《通俗易懂-深入理解TCP协议（下）：RTT、滑动窗口、拥塞处理》
《理论经典：TCP协议的3次握手与4次挥手过程详解》
《理论联系实际：Wireshark抓包分析TCP 3次握手、4次挥手过程》

4、TCP 异常关闭的研究测试

1服务器端只Recv消息而不Send消息

1.1 测试方法：

服务器程序在接受客户端的TCP连接后Sleep几秒钟，客户端程序在TCP连接后立即发送很多消息给对端后做相应动作（退出或等待），服务器程序Sleep完后开始Recv消息。

注意：服务器程序测试了Linux和Windows版本，但客户端只测试了Windows版本，如果是Linux客户端则有些Case的结果会不一样。

1.2 测试 Case：

- **1）客户端程序正常运行的情况下，拔掉网线，杀掉**

客户端程序：

目的：模拟客户端死机、系统突然重启、网线松动或网络不通等情况。

结论：这种情况下服务器程序没有检测到任何异常，并最后等待“超时”才断开TCP连接。

- **2）客户端程序发送很多数据包后正常关闭Socket并exit进程(或不退出进程)：**

  目的：模拟客户端发送完消息后正常退出的情况。

  结论：这种情况下服务器程序能够成功接收完所有消息，并最后收到“对端关闭”（Recv返回零）消息。

- **3）客户端程序发送很多数据包后不关闭Socket直接exit进程：**

  目的：模拟客户端程序退出而忘记关闭Socket的情况（比如通过Windows窗口的关闭图标退出进程，而没有捕获相应关闭事件做正常退出处理等）。

  结论：这种情况下服务器程序能够收到部分TCP消息，然后收到“104: Connection reset by peer”（Linux下）或“10054: An existing connection was forcibly closed by the remote host”（Windows下）错误。

- **4）客户端程序发送很多数据包的过程中直接Kill进程：**

  目的：模拟客户端程序崩溃或非正常方式结束进程（比如Linux下"kill -9"或Windows的任务管理器杀死进程）的情况。

  结论：这种情况下服务器程序很快收到“104: Connection reset by peer”（Linux下）或“10054: An existing connection was forcibly closed by the

remote host”（Windows下）错误。

2服务器端Recv消息并Send应答消息

2.1 测试方法：

服务器程序在接受客户端的TCP连接后Sleep几秒钟，客户端程序在TCP连接后立即发送很多消息给对端后做相应动作（退出或等待），服务器程序Sleep完后开始Recv和Send消息。

注意：服务器程序测试了Linux和Windows版本，但客户端只测试了Windows版本，如果是Linux客户端则有些Case的结果可能会不一样。

2.2 测试结果：

- **1）客户端程序发送很多数据包后正常关闭Socket并exit进程（或不退出进程）：**
  目的：模拟客户端正常关闭Socket后，服务器端在检查到TCP对端关闭前向客户端发送消息的情况。
  结论：这种情况下服务器程序接收和发送部分TCP消息后，在Send消息时产生“32: Broken pipe”（Linux下）或“10053: An established connection was aborted by the software in your host machine”（Windows下）错误。

- **2）客户端程序发送很多数据包后不关闭Socket直接exit或Kill进程：**
  目的：模拟客户端程序退出而忘记关闭Socket、或客户端程序崩溃或非正常方式结束进程的情况。
  结论：这种情况下服务器程序在Recv或Send消息时产生“104: Connection reset by peer”（Linux下）或“10054: An existing connection was forcibly closed by the remote host”（Windows下）错误。

3效果和总结

3.1 总结：

TCP发现网络异常（特别是Linux下的104错误或Windows下10054错误）的情况很多，比如网络本身的问题、中间路由器问题、网卡驱动器问题等不可抗拒因素，但下面是应用程序本身可能会导致的问题，也是我们需要进一步研究和解决的情况，特别是程序崩溃导致问题：

- **1）当TCP连接的进程在忘记关闭Socket而退出、程序崩溃、或非正常方式结束进程的情况下：**
  （Windows客户端），会导致TCP连接的对端进程产生“104: Connection reset by peer”（Linux下）或“10054: An existing connection was forcibly closed by the remote host”（Windows下）错误。
- **2）当TCP连接的进程机器发生死机、系统突然重**

启、网线松动或网络不通等情况下：
（Windows客户端），连接的对端进程可能检测不到任何异常，并最后等待“超时”才断开TCP连接。

- **3）当TCP连接的进程正常关闭Socket时，对端进程在检查到TCP关闭事件之前仍然向TCP发送消息：**
（Windows客户端），则在Send消息时会产生“32: Broken pipe”（Linux下）或“10053: An established connection was aborted by the software in your host machine”（Windows下）错误。

3.2 效果：

针对A项目的掉线问题，通过问卷调查和联系个别玩家等方法，发现掉线的情况很大部分是客户端程序直接退出了，因此推动项目组实现了客户端的Qos上报功能，最后通过客户端的Qos上报的统计数据得出客户端程序的崩溃比例比较高，占了总掉线的很大比率，当然其它情况也存在，但比例相对比较小。

因此，A项目首先应该解决客户端程序的崩溃问题，如果该问题解决了，也就解决大部分掉线问题。

5、TCP异常关闭的进一步研究测试

1问题背景

B项目游戏在跨服跳转时的掉线比例比较高，经过分析ProtocalHandler和tconnd的日志，发现掉线出现的情况是：tconnd发送了跨服跳转消息后立即关闭了Socket，客户端进程在接收到跨服跳转消息之前发送消息后收到Windows 10054错误，然后做断线重连失败。

B项目实现跨服跳转的流程是GameSvr给客户端程序下发的跨服跳转命令的同时携带了Stop请求，也就是说tconnd在向客户端转发跨服跳转消息后立即就会关闭当前的Socket连接，而且B项目的客户端程序会定期不断地向服务器上报消息。这又怎么会导致客户端程序收到10054错误而呢？鉴于此，对TCP的连接做进一步的场景测试分析。

2TCP异常进一步测试研究

2.1 测试方法：

客户端和服务器端程序建立TCP连接，服务器程序在TCP缓冲区中有消息或没有消息的情况下关闭Socket，客户端在对端Socket已经关闭的情况下继续Send和Recv消息。

注意：服务器端只测试了Linux版本，但客户端测试了Windows和Linux两个版本。

2.2 测试结果：

- **1）服务器端已经close了Socket，客户端再发送数据：**
  目的：测试在TCP对端进程已经关闭Socket时，本端进程还未检测到连接关闭的情况下继续向对端发送消息。
  结论：第一包可以发送成功，但第二包发送失败，错误码为“10053: An established connection was aborted by the software in your host machine”（Windows下）或“32: Broken pipe，同时收到SIGPIPE信号”（Linux下）错误。
- **2）服务器端发送数据到TCP后close了Socket，客户端再发送一包数据，然后接收消息：**
  目的：测试在TCP对端进程发送数据后关闭Socket，本端进程还未检测到连接关闭的情况下发送一包消息，接着接收消息。
  结论：客户端能够成功发送第一包数据（这会导致服务器端发送一个RST包 <已抓包验证>），客户端再去Recv时，对于Windows和Linux程序有如下不同的表现：
  - 2.1）Windows客户端程序：Recv失败，错误码为“10053: An established connection was aborted by the software in your host machine”；
  - 2.2）Linux客户端程序：能正常接收完所有消息包，最后收到正常的对端关闭消息（这一点与Window下不一样，RST包没有被提前接收到）。
- **3）服务器端在TCP的接收缓冲区中还有未接收数据的情况下close了Socket，客户端再收包：**
  目的：测试在TCP的接收缓冲区中还有未接收数据的情况下关闭Socket时，对端进程是否正常。

结论：这种情况服务器端就会向对端发送RST包，而不是正常的FIN包（已经抓包证明），这就会导致客户端提前（RST包比正常数据包先被收到）收到“10054: An existing connection was forcibly closed by the remote host”（Windows下）或“104: Connection reset by peer”（Linux下）错误。

3效果和总结

3.1 总结：

TCP应用程序某些看是正常的行为下也可能会导致对端接收到异常，比如当TCP接收缓冲区中还有未收数据的情况下关闭Socket，则会导致对端接收到异常关闭而不是正常关闭；反过来说，当TCP检测到异常关闭时并不一定表示业务上出问题了，因为很可能就是业务正常结束了。

下面是本次测试的主要结论：

- **1）**当TCP连接的对端进程已经关闭了Socket的情况下，本端进程再发送数据时，第一包可以发送成功（但会导致对端发送一个RST包过来）：之后如果再继续发送数据会失败，错误码为“10053: An established connection was aborted by the software in your host machine”（Windows下）或“32: Broken pipe，同时收到SIGPIPE信号”（Linux下）错误；之

后如果接收数据，则Windows下会报10053的错误，而Linux下则收到正常关闭消息；

- **2）**TCP连接的本端接收缓冲区中还有未接收数据的情况下close了Socket，则本端TCP会向对端发送RST包，而不是正常的FIN包，这就会导致对端进程提前（RST包比正常数据包先被收到）收到“10054: An existing connection was forcibly closed by the remote host”（Windows下）或“104: Connection reset by peer”（Linux下）错误。

3.2 效果：

B项目跨服跳转的掉线问题有相当一部分的种情况是tconnd向客户端转发跨服跳转消息后立即关闭Socket连接，而此时刚好客户端向tconnd发送了数据包：

- **第一种情况：**tconnd在关闭Socket的时刻其TCP的接收缓冲区中有未收的消息，这就使得tconnd进程的TCP向客户端发送的是RST包而不是正常结束的FIN包，所以客户端程序就会提前收到RST包（RST包会比正常数据提前收到），而不会先收完跨服跳转消息后再接收到正常结束消息，这就导致客户端收到网络异常断线而做重连，但之前的连接是tconnd主动关闭的，所以不可能重连成功，从而导致掉线；
- **第二种情况：**tconnd已经关闭了Socket后，客户端在接收到跳转消息和检测到TCP关闭之前向tconnd发送了消息，这就会导致客户端程序收到异常断线而做重连并失败。

最后，与B项目项目组一起讨论，改进了大部分跨服跳转的业务流程后，掉线比例j减少了很多，当然还是存在一定比例的掉线，但这应该就是其它一些原因了（网络异常问题原因很多，国内当前的网络环境下，掉线问题是不可能完全避免的）。

6、结束语

通常情况下，向TCP的Socket发送完数据后关闭Socket，大家认为这样很正常的方式肯定没有问题，对端应该正确收完数据后收到TCP的关闭消息，但实际上在某些情况下并非如此：当TCP本端的接收缓冲区中有未收的数据时关闭Socket，这会导致对端收到RST的异常关闭消息；当对端在本端已经关闭Socket的情况下再次发送消息时，也会导致对端收到异常关闭消息；还有为了避免TIME_WAIT而设置了SO_LINGER选项的话，也会导致连接提前夭折使对端收到RST异常关闭消息。

有些时候业务流程对是否引起掉线也很重要（特别是连接关闭流程），比如前面的B项目的跨服跳转掉线问题很大部分就是因为GameSvr请求关闭连接导致的。建议各个游戏项目的关闭流程（包括跨服跳转的原连接的关闭）最好都由客户端发起关闭，这样就一定程度上避免上述问题的发生（因为客服端发起关闭的时候，一般业务流程都走完了，服务器端也不会再向客户端发送消息了）。

程序收到网络异常的情况很多（最多的就是Linux下的104

错误和Windos下的10054/10053错误）：有网络本身的问题、也有应用使用不当的问题；有运营商之间的跨网络问题、网络中间路由器问题、玩家机器硬件（比如网卡及其驱动）问题和操作系统、杀毒软件、防火墙等软件问题，还有玩家的上网设备和路由器等中间设备问题等，但客户端程序崩溃问题有可能会占掉线的很高比例，这也是值得我们注意和改进的地方。还有种情况值得我们注意，有些TP-LINK的路由器，当UDP包大小超过其MTU值时会导致用户机器的网络断开，从而引起掉线（这个问题在某些项目的个别玩家中已经出现过）。

网络异常关闭引起掉线是当前游戏中普遍存在的问题，区别只在于掉线的比例多少，特别是国内各运营商之间跨网络访问更是不太顺畅，要将其完全消除是不可能的，但我们的目标是将其控制在较小的可接受范围内。

（原文链接：点此进入）

7、更多资料

《TCP/IP详解 - 第11章·UDP：用户数据报协议》
《TCP/IP详解 - 第17章·TCP：传输控制协议》
《TCP/IP详解 - 第18章·TCP连接的建立与终止》
《TCP/IP详解 - 第21章·TCP的超时与重传》
《技术往事：改变世界的TCP/IP协议（珍贵多图、手机慎点）》
《通俗易懂-深入理解TCP协议（上）：理论基础》
《通俗易懂-深入理解TCP协议（下）：RTT、滑动窗口、

拥塞处理》
《理论经典：TCP协议的3次握手与4次挥手过程详解》
《理论联系实际：Wireshark抓包分析TCP 3次握手、4次挥手过程》
《计算机网络通讯协议关系图（中文珍藏版）》
《UDP中一个包的大小最大能多大？》
《Java新一代网络编程模型AIO原理及Linux系统AIO介绍》
《NIO框架入门(一)：服务端基于Netty4的UDP双向通信Demo演示》
《NIO框架入门(二)：服务端基于MINA2的UDP双向通信Demo演示》
《NIO框架入门(三)：iOS与MINA2、Netty4的跨平台UDP双向通信实战》
《NIO框架入门(四)：Android与MINA2、Netty4的跨平台UDP双向通信实战》
《P2P技术详解(一)：NAT详解——详细原理、P2P简介》
《P2P技术详解(二)：P2P中的NAT穿越(打洞)方案详解》
《P2P技术详解(三)：P2P技术之STUN、TURN、ICE详解》
《高性能网络编程(一)：单台服务器并发TCP连接数到底可以有多少》
《高性能网络编程(二)：上一个10年，著名的C10K并发连接问题》
《高性能网络编程(三)：下一个10年，是时候考虑C10M并发问题了》
《高性能网络编程(四)：从C10K到C10M高性能网络应用的理论探索》
《不为人知的网络编程(一)：浅析TCP协议中的疑难杂症

(上篇)》

《不为人知的网络编程(二)：浅析TCP协议中的疑难杂症(下篇)》

《不为人知的网络编程(三)：关闭TCP连接时为什么会TIME_WAIT、CLOSE_WAIT》

>> 更多同类文章 ……