TCPCopy Manual

目 录

1 引言

XCopy 是由网易主导，多家公司成员[1]参与开发的具有在线 server 流量复制功能的一系列开源软件的总称，目的是复制在线 server 流量到测试系统中去，并引入在线的复杂性到测试系统，从而可以在测试系统中充分暴露在线的问题，帮助用户提前解决在线问题，降低上线失误率或者实现零失误。

XCopy 系列包括 TCPCopy、UDPCopy、MysqlCopy 等开源软件（这些软件都集成在 tcpcopy[2]开源项目内）。

XCopy 分为在线和离线工作方式，在线方式主要用来实时捕获在线请求数据包，离线方式主要从 pcap 格式的文件中读取在线请求数据包。由于离线方式依赖于抓包工具（如 *tcpdump*），而抓包工具在压力比较大的场合一般丢包非常严重，而且还会严重影响在线 IO，因此一般不推荐在高压情况下使用离线回放方式（以下内容一般不声明的话，默认是在线实时复制方式）。

XCopy 可以应用在以下领域：

1） 性能压力测试，如系统性能瓶颈查找、内核参数调优、不同程序性能比较等

2） 系统可用性测试

3） 冒烟测试

4） 回归测试，特别是针对技术类重构的测试

5） 。。。

TCPCopy 是 XCopy 系列中最重要的一个组件，其功能是复制在线 server 的 TCP 类型的请求数据包，修改 TCP/IP 头部信息，发送给测试服务器，达到欺骗测试服务器的 TCP 程序的目的，从而为欺骗测试服务器上面的上层应用打下基础。

由于互联网大部分应用是基于 TCP 的，并且其中的大部分协议是很容易被欺骗的，因此只需 TCPCopy 就能欺骗大部分上层协议，如 HTTP 协议、memcached 协议、POP3 协议、SMTP 协议、各种私有协议等等。

本文的结构如下：

第一章 引言

第二章 TCPCopy 原理及介绍

第三章 UDPCopy 原理及介绍

第四章 MysqlCopy 原理及介绍

[1] 目前成员包括淘宝，畅游等公司的相关人员和部分学生

[2] https://github.com/wangbin579/tcpcopy

2 TCPCopy 原理及介绍

2.1 原理

TCPCopy 包含两部分：TCPCopy client（*tcpcopy*） 和 TCPCopy server（*intercept*），其中，TCPCopy client 运行在在线系统，用来捕获在线请求数据包，而 TCPCopy server 运行在测试系统，用来做一系列辅助配合工作，如返回响应包头信息、维护路由信息等。

由于 TCP 交互是相互的，一般情况下需要知道测试服务器的响应数据包信息，才能利用在线请求数据包，构造出适合测试服务的请求数据包，因此基于数据包方式的在线模拟都需要测试服务的响应包的相关信息。

依据测试服务响应包信息的截获位置，TCPCopy 分为两种不同的架构，即传统架构（见图 2.1.1）和新架构（v0.8 及以上才支持，见图 2.1.2）。

采用图 2.1.1 中的传统架构，需要注意如下几个方面：

1）0.7.0 版本及之前的版本，只能采用传统架构，即在测试服务器端的 IP 层来截获响应包

2）0.7.0 以后的版本，默认采用图 2.1.1 中的方式来截获响应包

采用图 2.1.2 中的新架构：

1）0.8.0 及其以上版本，*configure* 的时候指定*--enable-advanced*，则采用图 2.1.2 中的方式来截获响应包

2）新架构需要设置路由，使得测试服务的响应包流向图 2.1.2 中的 assistant server。

具体介绍如下：

2.1.1 传统架构

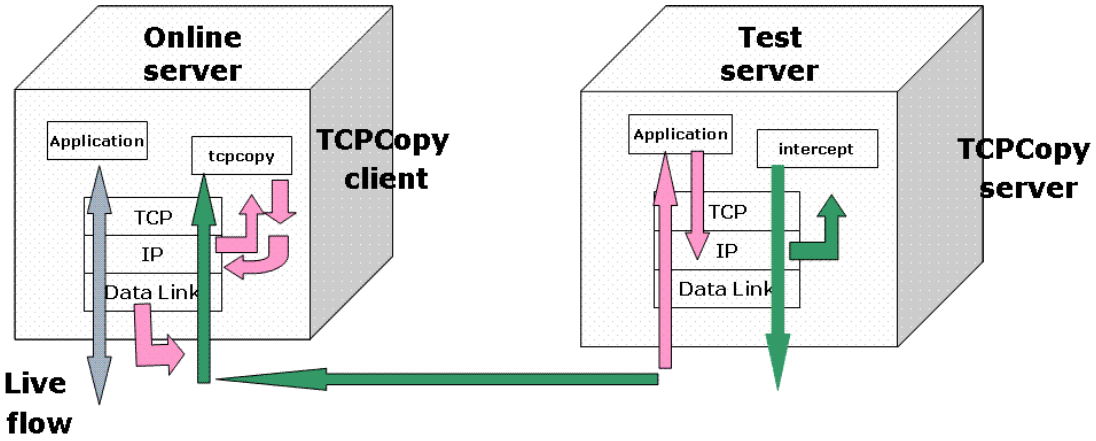

图 2.1.1 TCPCopy 传统架构原理图

在传统架构中，*intercept* 在 IP 层截获上层应用的响应包（如图 2.1.1 所示）。IP 层截获上层应用响应包的技术有很多，TCPCopy 采用了 netlink 技术（*intercept* 通过此接口获取响应包）与内核进行交互。根据 Linux 内核版本不同，可以采用 IP Queue 或者 NFQueue 来传递响应包给 *intercept*，具体如下：

内核版本<3.5， 默认采用 IP Queue 传递响应包给 *intercept*

内核版本>=3.5，默认采用 NFQueue 传递响应包给 *intercept*

当响应包传递给 *intercept* 后，我们提取响应包的必要信息（一般为 TCP/IP 头部信息），传递给 *tcpcopy*，同时通过 verdict 告诉内核，该如何处理这些响应包（如果没有设置白名单，这些响应包会在 IP 层被丢弃掉，从而无法被工作在数据链路层的 *tcpdump* 抓到）。

默认情况下，测试服务器的响应信息并不会返回给客户端，这样做的好处有：

1）不会增加出口带宽费用，节省测试成本

2）不会在互联网产生 ghost 数据包

3）不会干扰客户端的 TCP/IP 协议栈

传统架构的优点在于需要的测试服务器数量比较少，但也存在如下缺点：

1）性能瓶颈往往在 IP Queue 或者 NFQueue

2）*intercept* 扩展性不好，无法支持多进程响应包捕获操作

3）*intercept* 压力大时会影响测试服务器的测试结果

4）无法对测试服务器进行完整测试（没有覆盖到数据链路层的出口）

5）运维不方便

2.1.2 新架构（v0.8 及以上版本才支持）

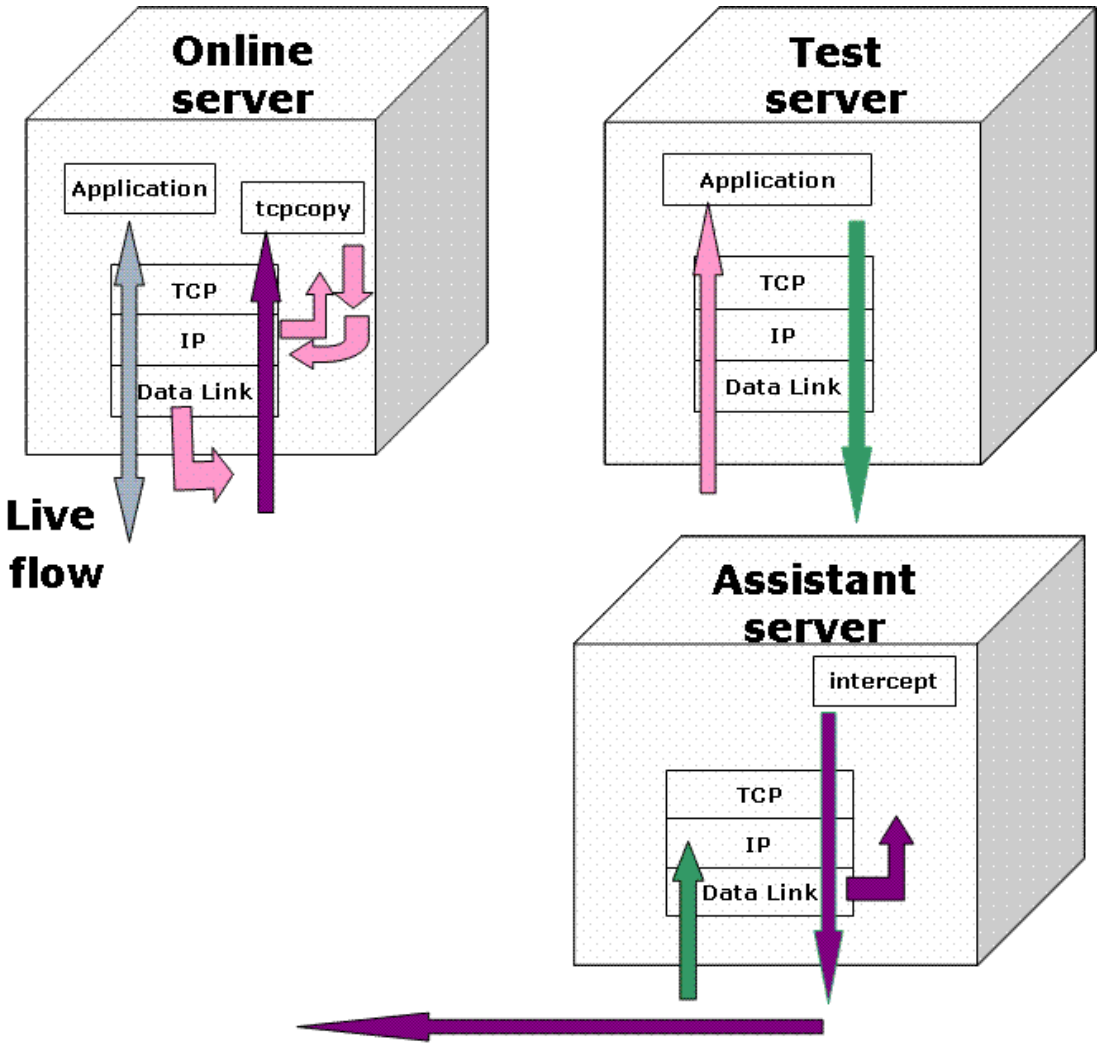

图 2.1.2 TCPCopy 新架构原理图

为支持极限测试，TCPCopy 除传统架构以外支持了一种新架构。如图 2.1.2 所示，新架构将 *intercept* 的工作从测试服务器（test server）offload 出来，放到另外一台独立的辅助服务器（assistant server，一般用同网段的一台闲置的服务器来充当辅助服务器）上面截获响应包，而且把传统架构从 IP 层捕获响应数据包的工作转移到从数据链路层抓响应包，这些改变大大降低了 TCPCopy 对测试机器的各种干扰（新架构下，除了路由设置，其它已经没有影响了），大大增强了 *intercept* 捕获响应包的能力，而且测试也更加真实。

具体如下：

在运行上层服务的测试服务器 test server 上设置路由信息，把待测试应用需要被捕获的响应数据包路由到辅助服务器 assistant server 上；在 assistant server 上，在数据链路层截获响应包[3]，从中抽取有用的信息，返回给相应的 *tcpcopy*。

[3] 在 assistant server 的 IP 层已经截获不到响应包了

新架构推荐采用 raw socket 方式来抓请求数据包（据测试，raw socket 抓请求数据包的丢包率远小于 pcap 抓请求数据包），而对于响应包的捕获，则推荐采用 pcap[4]（*--enable-pcap*）方式。

在进行高压测试时，需要在启动 *intercept* 的时候设置合适的 filter（通过-F 参数，类似 *tcpdump* 的 filter），达到多个实例共同完成抓响应包的工作。

对于 pcap 抓请求包的方式，理论上也可以设置合适的 filter，达到多个实例共同完成请求数据包的捕获工作。

相对于传统架构，新架构需要的机器资源会更多，使用起来更加复杂，需要了解 TCP、route 和 pcap filter 等相关知识，因此推荐有条件的并且熟悉上述知识的人使用新架构。

新架构总结如下：

优点：

1）更加真实

2）可扩展性更强

3）适合高并发场合

4）无 IP Queue 或者 NFQueue 的各种限制

5）对测试服务器几乎没有任何性能干扰的影响

6）运行服务的测试服务器上的运维更加方便

7）不会随运行服务的服务器崩溃而崩溃

缺点：

1）操作难度更大

2）需要的机器数量更多

3）需要的知识也更多

[4] 最好采用 pcap 1.0 以上版本

2.2 TCPCopy vs Tcpreplay

tcpdump+Tcpreplay 或者 *wireshark*+Tcpreplay 可以用来回放在线流量，这种方案可以解决 TCP 层以下的问题，如防火墙问题，然而，此方案仍有如下缺陷：

1）*tcpdump* 抓在线数据包，必然或多或少影响在线 IO，压力大的时候尤其明显

2）*tcpdump* 自身会丢包，特别是压力大的时候，丢包会很严重，远远超过 TCPCopy

3）*tcpdump* 抓的包需要保存下来，必然使其抓的包有限，一般很少抓几天的数据包

4）利用 Tcpreplay 重放，一般需修改数据包的源 IP 地址，这样必然跟在线环境不一样，比如对内核协议栈的冲击就很不一样

5）由于是离线回放，导致 Tcpreplay 回放的时候网络环境可能已经与抓包的时候不同了，而且 Tcpreplay 很难根据不同的环境做相应的调整，从而导致回放的效果与在线会有一定的差距

6）Tcpreplay 官方称不支持到服务端， 即回放对上层应用无效

7）Tcpreplay 相对 TCPCopy，使用更为复杂

因此，即使要测试 TCP 层以下的应用，也推荐使用 TCPCopy（加代理即可解决）。

TCPCopy 主要用来解决 TCP 层及其以上（如 http 协议）的流量复制问题，用于 server 的流量回放领域。总体来说，TCPCopy 有如下优点：

1）TCPCopy 能够对 server 进行回放，不仅可以离线回放，还可以实时回放

2）TCPCopy 实时复制在线请求数据包给测试服务器，并不需要大量 IO 操作，因此不影响在线 IO

3）实时回放所在的网络环境与当时的在线环境几乎是一样的

4）TCPCopy 默认情况下是不会去修改数据包的源 IP 地址

5）TCPCopy 实时复制转发过去的数据包，能够更好地继承在线数据包的网络延迟特征

6）TCPCopy 使用相对比较简单

2.3 影响 TCPCopy 的因素

可能影响到 TCPCopy 的地方有如下几个：

1） 抓包接口

2） 发包接口

3） 去往测试服务器的路上

4） 测试系统 OS

5） 后端应用

6） 传统架构下 Netlink 接口

我们分别介绍如下：

2.3.1 抓包接口

对于抓包，如果系统参数设置不合理，那么系统内核并不一定能把所有你关心的包都给你，压力越大，一般丢得越多，所以这方面相应的系统参数要设置好。

在线机器上面抓请求数据包，默认采用 raw socket input 接口，也可以采用 pcap 接口（*--enable-pcap*）。

如果采用新架构，那么响应包的捕获是通过抓包来实现的，这种情况下，推荐使用 pcap 抓响应包。

Pcap 抓包的好处大概有如下几点：

1）可以充分发挥 pcap 库的内核过滤功能，理论上能够极大地提高抓包的效率

2）利用 pcap 接口抓请求数据包的方式，据我们试验，其丢包率远大于 raw socket 抓请求数据包的方式，因此如果利用 pcap 抓请求数据包，最好配合 pfring（目前还不是很成熟，使用需谨慎），因为这能够大大降低丢包率，比 raw socket 方式丢包率更低。

2.3.2 发包接口

发包默认利用了 raw socket output 接口，还可以通过 pcap_inject（*--enable-dlinject*）来发包。压力比较大的场合推荐使用 pcap_inject 来发包，因为据我们某些场合的测试，合理利用 pcap_inject 来发包， tcpcopy 性能可以提升 30%以上。

需要注意的是，如果系统内核方面的系统参数设置不合理，那么系统并不一定能够成功发送所有包，特别是压力比较大的时候。

2.3.3 去往测试服务器的路上

首先说明一点：虚拟机环境下，跨机器等同于非虚拟环境下的跨网段；同机器等同于非虚拟环境下的同网段）。

tcpcopy 发送请求数据包以后，如果跨网段（虚拟机环境下，跨机器）的话，路途可能不会一帆风顺，因为被复制的数据包的源 IP 地址还是客户端的 IP 地址（这是为了引入在线复杂性，不会主动去改变数据包的源 IP 地址），可能会造成路途中的安全检测设备认为这些数据包是非法的或者是伪造的数据包，将 *tcpcopy* 转发的数据包给丢弃掉。这时候，在测试服务器端利用 *tcpdump* 来抓包，将不会抓到任何复制转发过来的请求数据包（包括三次握手的数据包等等）。你可以进一步在同一网段（虚拟机的环境下，同机器）试验一下，如果同网段（虚拟机的环境下，同机器）复制转发成功而跨网段（虚拟机的环境下，跨机器）复制转发不成功，那么跨网段（虚拟机的环境下，跨机器）转发出去的数据包应该在中途被丢弃掉了。

如果跨网段（虚拟机的环境下，跨机器）请求复制不过去，解决策略有两个：

1）先复制请求到本网段（虚拟机环境下，同机器）的应用代理，通过应用代理再把请求传递给跨网段的测试服务器（虚拟机环境下，跨机器）。

2）通过-c 参数指定 IP 地址列表，用以改变源 IP 地址为合法的 IP 地址（这些 IP 地址和在线机器的 IP 地址最好同属于同一个网段），这样就能够通过安全检测设备。

2.3.4 测试系统 OS

1）有些机器设置了 *rpfilter*（反向过滤技术），将会判断源 IP 地址是否是所信任的 IP 地址，如果不是，将会在 IP 层丢弃掉该数据包。因此在测试服务器端，如果利用 *tcpdump* 能够抓到转发过来的数据包，但利用 *netstat* 却看不到目标应用端口的任何 TCP 连接存在，那么很有可能系统设置了 *rpfilter*。

这种情况下，可以查看系统有没有设置 *rpfilter*，如果设置了，尝试去掉此设置后，利用 *netstat* 能不能检测到目标应用端口的 TCP 连接的存在（也就是这些被复制转发的数据包能否穿过测试服务器的 IP 层）。

2）系统只有内网网卡的情况下，对于外网请求数据包，IP 模块会认为这些数据包即使被处理了，也回不去，因此会丢弃掉这些来自外网的请求数据包。

这种情况下，一般需要安装外网网卡，才能解决此问题

3）传统架构下 *iptables* 的问题，如果全局 *iptables* 设置有冲突，会导致系统产生一系列诡异问题，比如，采用如下 iptables 命令来设置需要被截获的端口号：

```
iptables -A OUTPUT -p tcp --sport #port # -j QUEUE
```

采用 Append 的方式来增加 iptables 规则，容易被前面的规则所覆盖，导致设置无效。

4）在某些情况下，比如在极端情况下，如果你利用压力测试工具来发请求，也许压力测试工具已经测试完了，而转发的数据包中的第一个 syn 包还没有到达测试服务器，这样就造成原先串行的请求变成了并行请求，造成大量 syn 涌向测试服务器，这样就可能被测试服务器误报为 syn flood 攻击，一旦 syn 包被丢弃，那么相应的应用请求将无法传递给测试服务器(TCPCopy 针对三次握手的数据包，不会去重传)。

针对这种情况，要优化系统参数，才能有好的效果，或者你可以采用一些长连接的压力测试工具来发请求，比如 *httpsender*。

5）ip_conntrack 的问题，如果测试系统遇到“ip_conntrack: table full, dropping packet”的报警，那么需要先解决 ip_conntrack 问题后，再继续测试，否则可能丢失大量数据包。

可以参考如下方案来解决此问题：

https://github.com/wangbin579/tcpcopy/issues/42

6）针对传统架构，测试服务器内核中如果同时运行着 IP Queue 和 NFQueue，那么 IP Queue 模块就存在被损坏的可能。这种情况常见于在一切设置都合理的情况下，复制数据包给测试机器的时候，测试机器上面待测试的应用的连接只有 SYN_RECV 一种状态。

这时候可以参考如下方案：

https://github.com/wangbin579/tcpcopy/issues/60

2.3.5 测试服务器的应用程序

并不是每个请求过去，测试服务器上面的应用程序都能够及时处理或者都能处理，有可能触发了应用程序的 bug，导致请求迟迟没有响应，也有可能应用程序的协议不支持 *tcpcopy* 提前发送请求（类似 pipeline 形式的请求），仅仅处理 socket buffer 中的第一个请求，这些情况都会导致请求大量丢失。

2.3.6 传统架构下 Netlink socket 接口

下述内容只针对传统架构，而且采用默认 IP Queue 模式的情况下有效。

假设后端响应都顺利，压力比较大的时候，IP Queue 模块传递响应包给 netlink socket 的时候，也会遇到丢包现象，这时候可以通过 *cat /proc/net/ip_queue* 命令，查看 IP Queue 运行情况：

1） 如果 *Queue dropped* 的数值不断增大，则需要修改 ip_queue_maxlen 参数，比如：

echo 4096 > /proc/sys/net/ipv4/ip_queue_maxlen

2） 如果 *Netlink dropped* 的数值不断增大，修改 rmem_max 和 wmem_max 参数，比如：

sysctl -w net.core.rmem_max=16777216

sysctl -w net.core.wmem_max=16777216

3）如果 IP Queue 相关系统参数的值不管设置为多大，*Netlink dropped* 还是在不断增大，可以尝试重启 IP Queue 模块：

modprobe -r ip_queue

modprobe ip_queue

或者采用新架构

2.4 TCPCopy 优化

要让 TCPCopy 更好地工作，需要在各个方面进行优化（特别是系统参数方面），除了上面所讲的影响 TCPCopy 的地方外，我们还可以进行如下方面的优化。

我们主要提供的优化参数如下：

1）*tcpcopy* 的-t 参数，默认 120s（0.9.0 以下版本为 60s），如果后端普遍响应很慢或者是长连接应用的场合，建议设置更大一些

2）*tcpcopy* 的-m 参数，此参数一般只有在内核版本>=2.6.32 才有效，默认 512M，*tcpcopy* 所占内存一旦超过此阈值，就会退出。如果内存需要超过此值，请设置合理的值

3）如果仅仅复制在线的一台流量（比如单台在线流量很大）给一台测试服务器，那么推荐使用 single 模式的 TCPCopy 版本，这个模式的版本在 *intercept* 程序中不会保留路由信息，因此 *intercept* 性能会更高一些

4）如果复制的在线压力非常大，除了调整系统参数以外，还需要调大 *tcpcopy* 的-C 参数，以增加 *tcpcopy* 和 *intercept* 的通信连接数（默认为 3 个连接），从而增加吞吐量，比如采用 8 条连接支持响应包的大量返回。

2.5 TCPCopy 不足的地方

1）TCPCopy 的性能跟所处理的包的多少关系很大

2）无法保持不同连接的请求的有序性

3）对于传统架构，*intercept* 过分依赖于内核，需系统参数调优才能有好效果

4）TCPCopy 会消耗一定的带宽，主要在转发在线请求数据包方面

5）TCPCopy 文档严重缺乏，有时候还无法及时更新文档，在此表示抱歉

6）最新文档只对最新 TCPCopy 版本生效

2.6 TCPCopy CHANGES

1）2011.09 发布 TCPCopy 0.1 版本

2）2011.11 发布 0.2 版本，解决若干 bug

3）2011.12 发布 0.3 版本，支持 mysql 请求复制

4）2012.04 发布 0.3.5 版本，新增多重复制功能

5）2012.05 发布 0.4 版本，解决若干 bug

6）2012.07 发布 0.5 版本，重构代码，解决大文件下载和上传的复制问题，
解决抓包超过 MTU 的问题

7）2012.08 发布 0.6 版本，新增离线回放功能

8）2012.10 发布 0.6.1 版本，支持 *intercept* 多线程模式

9）2012.11 发布 0.6.3 版本, 解决 fast retransmitting 问题

10）2012.11 发布 0.6.5 版本, 支持 *NFQueue*（针对 Linux 高版本内核）

11）2013.03 发布 0.7.0 版本，支持 single 模式、DR 模式（复制给带有 LVS 的子系统），
以及支持响应延迟控制的 paper 模式，并解决 mysql 的若干 bug。

12）2013.06，发布 0.8.0 版本，支持新架构，合并响应包，多连接支持，优化 *intercept*，
离线回放加速，去除 *intercept* 的多线程模式，解决短连接的若干问题

13）2013.08，发布 0.9.0 版本，支持 *tcpcopy* 从数据链路层发包，去除 GPL2 协议代码，
支持改变请求数据包的源 IP 地址，同步 TCP 数据包中的 timestamp 信息，
修正若干个 bug

14）2013.09，发布 0.9.3 版本，0.9.0 的增强版本

3 UDPCopy 原理及介绍

基于 UDP 的复制很简单，一般直接改变 UDP 头部的目的 IP 地址和目的端口，如果数据包的长度超过 MTU，进行 IP 分片即可。

UDPCopy 默认情况下是实时复制版本，由于从 IP 层末尾抓包，避免了组装 IP 分片的问题，大大简化了程序的设计；对于离线模式的版本，由于抓包一般都是从数据链路层抓包的，所以遇到 IP 分片后的数据包，目前为止还不能正常处理。

4 MysqlCopy 原理及介绍

MysqlCopy 是建立在 TCPCopy 基础上的，针对 mysql 应用的请求复制工具。

MysqlCopy 原先所依据的协议文档地址如下：
http://forge.mysql.com/wiki/MySQL_Internals_ClientServer_Protocol
由于上述地址已经无效，最新的协议文档如下：
http://dev.mysql.com/doc/internals/en/client-server-protocol.html

这里需要注意的是 MysqlCopy 只支持 4.1 and later，不支持之前的老版本

由于 mysql 协议是有状态的，mysql 协议与其它无状态的协议有如下不同：

1） 中途截获 mysql 请求，一般无法构造有效请求，因为缺乏前面的有效信息
2） Mysql 会话需要登入过程
3） 如果测试服务器的 mysql 关闭掉会话过程，而在线还在会话，这时候重新建立会话的过程中，还需要重新发送登入过程的 packets 和 prepare 语句的 packets（如果之前会话有 prepare 语句的话）

正因为这些特征，导致了其处理的复杂性，为了降低复杂度，MysqlCopy 分为两种工作方式：

1） skip-grant-tables 工作方式
2) 正常 mysql 工作方式

在线服务器上面的 mysql 程序采用正常的工作方式，测试服务器上面的 mysql 程序可以采用 skip-grant-tables 模式或者正常模式。如果测试服务器采用 skip-grant-tables 来运行 mysql，那么 MysqlCopy 在 configure 的时候需要增加 *--enable-mysqlsgt*; 如果测试服务器上面的 mysql 程序跟在线一样，采用正常的模式运行，那么 MysqlCopy 在 configure 的时候需要增加 *--enable-mysql*。这两种模式所不同的是正常模式需要提供用户名和密码，而且源代码也略微不一样（为了性能考虑，目前采用了从 configure 来区分两种工作方式）。

下面讲述 mysql 协议登入过程的原理：

采用 skip-grant-tables 模式，无需 *intercept* 传回 crypted password，所以只需传递 TCP 和 IP header 给 *tcpcopy* 即可完成 mysql 协议的正常交互，而 mysql 正常模式下，需要传递 server 端的 crypted password 给 *tcpcopy* 以便 *tcpcopy* 来根据 crypted password 修改 mysql 协议的登

入过程的数据包的内容，才能欺骗测试服务器的 mysql。

最后，对 mysql 进行流量复制时，需要注意如下事项：

1）如果测试服务器的 mysql 需要工作在 skip-grant-tables 模式下，configure 的时候增加 *--enable-mysqlsgt*，编译后的代码仅仅适用于这种工作方式

2）如果采用正常模式，也即非 skip-grant-tables 模式，configure 的时候需增加 *--enable-mysql*，编译后代码仅仅适用于正常模式；执行的时候需要设置-u 参数，也即用户名密码，用户名必须跟在线一致，密码可以不一样，而且最重要的是这些用户在测试系统 user 表中的配置信息，必须要跟在线系统一致（比如测试走默认的“%”，而在线走非默认的“%”，这样配置就不一样，会导致登入过程校验不一样，从而复制失效）。

具体执行命令举例如下（采用 TCPCopy 0.6.3+版本）：

./tcpcopy -x 3306-xxx.xxx.xxx.xxx:3306 -u user1@password1,user2@password2 -d

3）如果采用 sysbench 来测试 MysqlCopy，由于 sysbench 的连接可能是永久连接，所以要求 *tcpcopy* 运行在 sysbench 启动之前；如果复制在线系统的 mysql 请求，MysqlCopy 需要捕获到新建连接才能开始工作，所以最好能够经常杀连接或者重启应用，使其重新建立到 mysql 的连接，这样就能够被 MysqlCopy 捕获到登入信息和 prepare statement 语句

4）正常模式的 mysql 请求复制，会比 skip-grant-tables 方式传递更多的响应内容，所以推荐后一种方式

5 TCPCopy 用法参考

5.1 How to configure

我们可以在 *configure* 的时候指定 TCPCopy 的运行模式，我们针对传统架构和新架构，分别介绍如下：

5.1.1 传统架构

1) *./configure*

经过编译后，TCPCopy 工作方式为实时复制模式。

值得注意的是，在内核 3.5 之前，*intercept* 默认采用 IP Queue 模块；在内核>=3.5 后，*intercept* 默认采用 NFQueue 模块（因为 3.5 以后就不支持 IP Queue 了）

2) *./configure --enable-nfqueue*

经过编译后，*intercept* 采用 NFQueue （内核版本 3.5 以后默认采用 NFQueue），实时复制模式

3) *./configure --enable-single*

经过编译后，TCPCopy 工作在非分布式模式，只能复制一台在线流量到测试服务器（即 *tcpcopy* 单实例），*intercept* 不会维护路由信息，以提高单台机器复制的潜能，适合于复制单台在线机器大流量应用的场合或者短连接应用的场合

4）*./configure --enable-offline*

经过编译后，工作在离线回放方式，适合于不方便在在线服务器上面实时复制流量的场合或者虚拟机环境下的场合。

这里需要注意如下内容：

a）离线回放所依据的 pcap 文件，最好只包含在线请求数据包，不要包含在线的响应包，这样做的好处就是减少 pcap 文件所占大小，而且会提升离线回放程序的性能

b）抓包生成 pcap 文件的时候，一定要确保不要丢包（可以采用 pfring 来抓包）。

c）离线模式需要有-i 参数，用来指定 pcap 文件所在的路径

d）离线回放由于存在着定时问题，所以回放的结果与真实在线的结果会有一定的差距

5）*./configure --enable-dr*

经过编译后，工作在 DR 模式。DR 模式取之于 LVS 的 DR 应用场景，是为了解决复制在线流量给带有 LVS(由于大部分场景下采用 DR)的子系统。

由于在线流量复制给 LVS 后，LVS 会把不同连接的数据包按照一定的策略分发给不同的测试服务器，导致了与常规 TCPCopy 用法的不同。

下面内容讲述的是在传统架构下的 DR 模式相关内容。

为了解决响应包如何返回的问题，此模式下的 *tcpcopy* 会把每个会话的路由信息传递给每一台测试服务器上面相应的 *intercept*。因此 *tcpcopy* 执行命令的时候会多出-s 参数，用来指定真正的 real server 的 IP 地址，而 *intercept* 则需要设置-x 参数，参数的值就是 LVS 服务器的所用的实际 IP 地址，设置的目地是让 LVS 发出的数据包（即检测 real server 的健康程度的数据包）能够通过测试服务器的 IP 层，这样 LVS 才会认为这些 real server 是健康的，而不是处于崩溃的状态。

理论上，如果数据包到达测试服务器之前，其目的 IP 地址存在 NAT 变换（Destination Network Address Translation.），也就是说目的 IP 地址会被改变的情况下，那么 DR 模式也能适用，这种场景下的应用可以看成是 DR 的一种特殊应用场景。

值得注意的是，默认情况下，如果 LVS 子系统中的一台 real server 机器上面的 *intercept* 进程崩溃后，仅仅重新启动 *intercept*， TCPCopy 不会正常工作，需要重新启动所有 *tcpcopy* 才能顺利工作，为了避免此麻烦， *tcpcopy* 运行的时候需要加上-L 参数即可解决此问题。

6）*./configure --enable-*rlantency

默认情况下，响应数据包的网络延迟是不保持的，此模式加入了响应包的返回延迟，目的是更加逼真地模拟在线的情况。

7）*./configure --enable-pcap*

这种模式是为了能够利用 pcap 库（1.0 版本及其以后）进行抓包，利用 pcap 的内核过滤接口，理论上可以提升抓请求数据包的效率，比如当在线机器应用种类比较多，需要复制某一个应用的请求到测试系统的时候，这种方式效率会比较高。

8）*./configure --enable-combined*

这种模式是合并响应包，降低系统调用，为高性能而准备的。

0.8.0 及其以上版本默认是开启合并模式，因此 0.8.0 版本将不再兼容之前的版本，因此 0.8.0 之前的 *intercept* 和 0.8.0 版本的 *tcpcopy* 不能够通信或者 0.8.0 版本的 *intercept* 和 0.8.0 之前的 *tcpcopy* 不能够通信。

注意：这种模式只有在 0.8.0 版本或者以上版本才有效

9）*./configure --enable-dlinject*

此模式是为了支持 *tcpcopy* 能够从数据链路层发送请求数据包。

从数据链路层发包的好处是不会去干扰在线服务器的 IP 模块（比如不会去干扰

ip_conntrack 模块），但不好的地方是需要自己去解决路由问题。

10）*./configure --enable-debug*

这种模式是为了输出 debug 日志，方便问题诊断的，如果你遇到了无法解决的问题，请采用此模式运行 TCPCopy（一般运行 1 分钟即可）

需要注意的是， *tcpcopy* 和 *intercept* 程序不仅要求采用同一版本的程序，而且一般需要采用同样或者兼容的 configure 配置，以防诡异问题产生。

5.1.2 新架构

对于新架构，configure 的时候必须显式指定--enable-advanced，除此以外，常用的 options 选项如下：

1）*--enable-pcap*

这种模式编译 *tcpcopy* 和 *intercept*，是为了能够利用 pcap 库进行抓请求数据包和响应数据包，利用 pcap 的内核过滤接口，理论上可以提升抓包的效率，比如当应用种类比较多，需要捕获某一个应用的响应包，这种方式效率会比较高。

2）*--enable-dr*

新架构下自动启用这种模式，目的是为了支持 *tcpcopy* 能够跟任意一台机器上面的任意一个 *intercept* 实例进行通信。

3）*--enable-dlinject*

此模式是为了支持 *tcpcopy* 能够从数据链路层发送请求数据包。

从数据链路层发包的好处是不会去干扰在线服务器的 IP 模块（比如不会去干扰 ip_conntrack 模块），但不好的地方是需要自己去解决路由问题。

在压力比较大的场合，推荐使用此模式。

为了解决新架构下响应包返回的问题， 需要在每一台运行服务的测试服务器上面设置路由信息，让需要返回的响应包路由到图 2.1.2 中的 assistant server 上面，在 assistant server 服务器上面进行响应包的捕获，因此新架构下，*intercept* 无需在运行服务的测试服务器上面都部署运行，简化了整个 TCPCopy 架构。

5.1.3 总结

传统架构一般能够满足大部分在线需求，而对于流量巨大的场合，最好采用新架构。

最后，不管采用何种架构，需要注意的是，0.8.0 版本及其以后，默认采用了合并响应包方式，所以和之前的版本不兼容，所以最好 *tcpcopy* 和 *intercept* 采用同样或者兼容的版本

5.2 How to run

5.2.1 传统架构

在传统架构下，一般 TCPCopy 执行命令如下：

在测试服务器 (需要 root 用户权限):

采用 IP Queue 模块（内核<3.5，默认采用 IP Queue）:

1) *# modprobe ip_queue # if not running*

2) *# iptables -I OUTPUT -p tcp --sport <port> -j QUEUE # if not set*

3) *# ./intercept*

Or

采用 NFQueue 模块（内核>=3.5，默认采用 NFQueue）:

1) *# iptables -I OUTPUT -p tcp --sport <port> -j NFQUEUE # if not set*

2) *# ./intercept*

这里需要注意，*iptables* 命令中的 port 是变量，应根据具体应用项目而定。

在在线服务器端：

 # ./tcpcopy -x localServerPort-targetServerIP:targetServerPort

具体地，可能还需要加入其他参数，后面会讲述 TCPCopy 的常用参数。

5.2.2 新架构

由于使用比较复杂，请参考如下具体例子：

1) TCPCopy 新架构使用方法

http://blog.csdn.net/wangbin579/article/details/8950282

2) TCPCopy 新架构具体复杂应用实例

http://blog.csdn.net/wangbin579/article/details/8994601

3) 如何利用 pcap 接口从数据链路层发包？

http://blog.csdn.net/wangbin579/article/details/10148247

5.3 TCPCopy client

tcpcopy -h 可以得到帮助文档，不同模式，命令的个数可能有所不同。

下面详细介绍常用参数设置

-x 参数

格式：-x <transfer,>

Transfer 具体格式如下：

服务器对外 IP 地址:服务器应用端口号-测试服务器 IP 地址:测试服务器应用端口

或者

服务器应用端口号-测试服务器 IP 地址:测试服务器应用端口

Transfer 之间用“,”隔开，IP 地址和端口号之间用“:”隔开，服务器应用端口号和测试服务器 IP 地址之间用‘-’隔开

举例：

./tcpcopy -x 80-192.168.0.2:18080

复制在线机器的 80 端口应用的请求到 192.168.0.2 上面的 18080 端口

如果 configure 有*--enable-dlinject* 选项，也即从数据链路层发送数据包，那么还可以这样来执行：

服务器对外 IP 地址:服务器应用端口号@本地出去的 mac 地址-测试服务器 IP 地址:测试服务器应用端口@下一个跳转的 mac 地址

或者

服务器应用端口号@本地出去的 mac 地址-测试服务器 IP 地址:测试服务器应用端口@下一个跳转的 mac 地址

这里的 mac 地址是为了填补数据包 frame 的头部信息，以解决路由问题。

-i 参数

在离线模式下

格式：-i <file>

其中 file 是 pcap 文件的文件路径

在 pcap 模式下

格式：-i < device,>

指定从哪个或者哪些网卡设备上抓包

举例：

-i eth0

从 eth0 设备上来抓包

-o 参数

格式：-o < device,>

指定从哪个网卡设备上发包

举例：

-o eth0

从 eth0 设备上来发包

需注意如下事项：

1）此参数只有在*--enable-dlinject* 模式下才有效

2）-o 参数需要设置成与转发 IP 地址相匹配的网卡设备

比如：转发 IP 地址为外网 IP 地址，那么-o 参数就设置成外网网卡设备的名称

-I 参数

离线模式下，降低请求之间的间隔，对稀疏的请求访问，其加速非常有效果

举例：

./tcpcopy -x 80-192.168.0.2:8080 -I 1000 -i online.pcap

对请求之间间隔 1000ms 以外的请求进行加速

注意只有在离线模式下有效

-a 参数

离线模式下，对请求数据包的访问进行加速

举例：

假设 online.pcap 文件为在线请求数据包的抓包文件，时间间隔为 60 分钟

./tcpcopy -x 80-192.168.0.2:8080 -a 2 -i online.pcap

执行此命令后，离线回放加速了 2 倍，只需要 30 分钟，离线回放就能完成

需要注意的是，此命令只有在离线模式下才有效，而且-a 参数设置越大，丢请求的概率也越大。

-B 参数

格式：-B < num>

指定 pcap 抓请求数据包的缓冲区大小，默认为 16（单位为 M）

需要注意的是此参数只有在 pcap 模式下才有效（*configure --enable-pcap* 的情况下）

-F 参数

格式：-F < filter>

指定抓包的过滤条件，具体格式可以参考 *tcpdump* 或者 pcap filter 的格式

一旦设置此参数，*tcpcopy* 不会自动去构造 filter，采用-F 参数指定的 filter 去过滤请求数据包

举例：

-F 'tcp and dst port 11311 and dst host 10.100.10.1'

复制目的地址为 10.100.10.1 且目的端口为 11311 的 TCP 请求数据包

需要注意的是此参数只有在 pcap 模式下才有效（*configure --enable-pcap* 的情况下）

-C 参数

指定 *tcpcopy* 和 *intercept* 之间的连接数量，默认为 3 条连接供响应包的返回和路由信息的传递，在高压情况下，加大此参数，可以提高吞吐量，但 select 性能可能会有一定程度的下降。

-c 参数

格式：-c <IP,>

改变请求数据包的源 IP 地址为 IP 地址列表中的某个 IP 地址

举例:

假设你复制的请求是本地 localhost(127.0.0.1)的请求（比如抓包得到的结果类似于 127.0.0.1:80→127.0.0.1:8080），那么直接复制这样的请求到其它机器上去，一般是不行的，需要修改数据包的源 IP 地址，-c 参数就是用来改变源 IP 地址的。

./tcpcopy -x 8080-192.168.0.2:8080 -c 192.168.0.1

复制 127.0.0.1 上面的 8080 端口应用的请求到 192.168.0.2 上面的 8080 端口，同时修改源 IP 地址 127.0.0.1 为 192.168.0.1 地址

需要注意如下内容：

1）此参数在跨网段安全性比较高的场合，比较有效。

2）*tcpcopy* 仅仅改变客户端 IP，不会自动解决改变所带来的端口冲突问题。

3）当客户端 IP 地址较多时，为了尽可能地避免端口冲突问题，IP 地址列表的数目一般越多越好。

-n 参数

如果你要进行多重复制，那么此参数的值就是代表复制过去的流量是在线的 n 倍，倍数越小，效果越好，因为多重复制的原理是修改端口号，因此复制的倍数越大，端口冲突的概率越大，特别是源 IP 地址非常少，短连接的的内网应用场合。系统默认最大值为 1023 倍。

举例

./tcpcopy -x 80-192.168.0.2:8080 -n 3

复制 3 倍的在线服务器的 80 端口应用请求流量到 192.168.0.2 的 8080 端口

-f 参数

如果你要运行多个 *tcpcopy* 实例，复制请求到同一台测试服务器上面去，那么此参数就是为此设置的，常见于逐步增大流量的场合。最大值为 1023。

举例：

./tcpcopy -x 80-192.168.0.2:8080

./tcpcopy -x 80-192.168.0.2:8080 -f 1

./tcpcopy -x 80-192.168.0.2:8080 -f 2

这里我们在在线服务器运行三个实例，可以把测试服务器的流量放大到在线的 3 倍。

需要注意的是，如果要用-f 参数，复制给同一台测试服务器上面的同一个应用的不 同 *tcpcopy* 实例之间的-f 参数的值必须是不同的。

-r 参数

如果你想复制在线服务器应用的部分流量，可以采用-r 参数来实现，参数范围是 1～99, 其它值都是全流量复制。

举例：

./tcpcopy -x 80-192.168.0.2:8080 -r 20

这里 *tcpcopy* 复制在线服务器 8080 端口应用的 20%流量给后端服务器，需要注意的是 20%是根据 session（这里 session 是由客户端 IP，客户端端口决定）来统计的。

-r 参数常见于对测试应用进行 profile 的场合或者测试服务器配置不如在线服务器的场合。

-m 参数

此参数一般只有在 Linux 内核 2.6.32 版本及其以后版本才有效，如果内存超过了此设置值，那么 *tcpcopy* 就自动退出，默认是 512M，这是为了保护在线，防止 *tcpcopy* 占用过多内存。

-s 参数

格式：-s <server,> intercept server list

server 具体格式如下：

ip_addr1:port1, ip_addr2:port2, ...

指定真正运行 *intercept* 的地址列表，通过此参数可以和任意一台服务器的任意一个 *intercept* 实例进行通信。

举例如下：

./tcpcopy -x 80-10.120.12.211:28080 -s 10.120.12.161:36525

复制在线 80 端口的流量到 IP 地址为 10.120.12.211 的测试子系统中去（其中测试系统上面运行 28080 端口服务），并设置 *tcpcopy* 与 *intercept* 的通信地址列表（*tcpcopy* 与 10.120.12.161 上面的 36525 端口的 *intercept* 通信）。

注意：

1） -s 参数只有 *configure --enable-dr* 或者新架构下才有效

2） lvs 场景如何使用 tcpcopy，请参考：

http://blog.csdn.net/wangbin579/article/details/8612952

-t 参数

如果你的在线应用响应非常慢或者长连接应用的场合，那么推荐设置更大的-t 参数值（默认是 120s，如果 120s 内没有收到测试服务器的响应，那么这个请求会话就可能被 *tcpcopy* 给丢弃掉）。

-l 参数

设置错误日志文件的路径

-p 参数

格式:-p <num> remote server listening port

远程 *intercept* 的监听端口，默认是 36524。

注意事项：

在 *configure --enable-dr* 或者新架构下，如果 *tcpcopy* 在-s 参数中仅仅设置了 IP 地址，那么端口就采用默认的-p 参数值；如果-s 参数设置了端口号，那么系统将替换掉-p 参数所设置的端口值

-P 参数

格式：-P <file>

save PID in <file>, only used with -d option

-d 参数

设置 *tcpcopy* 以 daemon 运行

5.4 TCPCopy server

可以通过 *intercept -h* 来查看 *intercept* 命令的帮助文档

-x 参数

格式：-x <passlist,>

此参数只有在传统架构下才有效。

passlist 指通过 *intercept* 的 IP 地址列表，IP 地址之间以“,”隔开，这个参数的作用是指这些客户端 IP 地址的访问，其响应将返回给客户端，不会被 *intercept* 丢弃掉，常见于需要管理测试服务器应用的场合或者其它应用需要访问测试服务器被测应用的场合。

举例

./intercept -x 192.168.0.3,192.168.0.4

从 192.168.0.3 或者 192.168.0.4 发出的请求，其响应并不会被测试服务器 drop 掉。

-p 参数

设置监听端口，默认为 36524

需要注意的是，一旦改了这个默认端口，*tcpcopy* 端必须调整端口值

-i 参数

格式：-i < device,>

指定从哪个网卡设备上抓响应包

需要注意的是此参数只有在新架构 pcap 模式下才有效

-F 参数

格式：-F < filter>

指定抓响应包的过滤条件，具体格式可以参考 *tcpdump* 或者 pcap filter 的格式

举例：

./intercept -F 'tcp and src port 11511' -d

指定 *intercept* 抓源端口为 11511 的 TCP 响应包

需要注意的是此参数只有在新架构 pcap 模式下才有效

-o 参数

格式：-o <target>

指定抓响应包的过滤条件，此参数只有在非 pcap 模式下才有效果,而 pcap 模式无需设置此参数。

举例：

./intercept -o xxx.xxx.xxx.161:18080

intercept 抓源 IP 地址为 xxx.xxx.xxx.161，源端口号为 18080 的响应包

-b 参数

后跟 IP 地址，用来设置允许监听的 IP 地址，常见于安全性比较高的场合。默认情况下，监听 IP 地址默认为*，也即这台机器的所有 IP 能够监听 36524 端口，如果不允许外网 IP 地址访问，可设置内网 IP 地址，这样就只允许内网 IP 地址监听这台测试服务器的 36524 端口。

例如

./intercept -b 192.168.0.1

只有 192.168.0.1 才能监听 36524 端口，目的地址是其它 IP 地址的访问，均会被 OS 拒绝。

-l 参数

设置错误日志文件的文件路径

-P <file>

save PID in <file>, only used with -d option

-d 参数

设置 *intercept* 以 daemon 运行

6 UDPCopy 用法参考

旧版本：

代码地址：https://github.com/wangbin579/udpcopy
旧版本只支持传统架构。

一般操作如下：
在在线服务器端 (需要 root 用户权限):

./udpcopy -x local_port-remote_ip:remote_port

在测试服务器端，设置 iptables 命令 drop 掉响应信息，如果 UDP 应用程序有响应的话。
设置 iptables 命令格式如下：
iptables -I OUTPUT -p udp --sport <port> -j QUEUE （假设传统架构下采用 IP Queue）
需要注意的是，这里 port 是变量，随具体应用端口而定

新版本：

代码地址：https://github.com/wangbin579/tcpcopy
新版本目前只针对传统架构进行了测试。
以下内容只针对传统架构。

在 tcpcopy 0.9.0 版本，udpcopy 已经集成到 tcpcopy 中去，以后不再维护旧版本。
在在线服务器端 (需要 root 用户权限):

./tcpcopy -x local_port-remote_ip:remote_port

在测试服务器端，需要运行 *intercept* 程序（为未来考虑），而且如果 UDP 应用程序有响应的话，需要设置 iptables 命令 drop 掉响应信息。
设置 iptables 命令格式如下：
iptables -I OUTPUT -p udp --sport <port> -j QUEUE （假设传统架构下采用 IP Queue）
需要注意的是，这里 port 是变量，随具体应用端口而定

7 MysqlCopy 用法参考

除了 TCPCopy 的 *configure* 命令选项外，要复制 mysql 请求，必须在 *configure* 的时候指定工作模式：

./configure -- enable-mysql ---mysql 模式

./configure -- enable-mysqlsgt ---mysql skip-grant-tables 模式

7.1 Mysql 正常模式

需要指定用户名密码对，举例如下：

./tcpcopy -x 3306-xxx.xxx.xxx.xxx:3306 -u user1@password1,user2@password2 -d

这里复制在线的 3306 端口请求，也即复制 mysql 请求到 IP 地址为 xxx.xxx.xxx.xxx 机器的 3306 端口中去，其中指定用户名 user1,密码为 password1;用户名 user2，密码 password2。用户名 user1 和 user2 在测试服务器和在线机器上面的 user 表中的信息必须一致，密码可以不一样。

需要注意的是，TCPCopy 0.6.3 以下版本针对多个用户名之间的间隔符采用了冒号“:”，具体是哪一种，可以查看 *tcpcopy -h* 命令

7.2 Mysql skip-grant-tables 模式

这种工作方式是推荐的工作方式，因为简单，无需强制要求用户的权限一样，无需提供密码，唯一要求就是要捕获到新的连接才能工作。

命令很简单：

./tcpcopy -x 3306-10.130.12.161:3306 -d

8 QA

1） 事件机制为什么不用性能更高的 epoll

答：首先 epoll 并不一定总是性能很高，其次 fd 数量一般情况下极少，因此采用 select 更适合这种应用场合，这样既简单，性能可能还更高（TODO 性能方面需要确认）。

2） 为什么默认不用 libpcap 库来抓包

答：为了方便用户安装 TCPCopy，并不是每个 OS 都默认安装 libpcap 开发库的。

如果想用的话，*configure* 的时候指定*--enable-pcap* 即可。

3） 为什么需要 TCPCopy server（*intercept*）

答：这是因为需要它来传递响应包头信息，这样才能完成 TCP 交互，还有默认情况下，TCPCopy server 还解决响应包头如何返回的问题和响应包合并的工作。

值得注意的是，针对 mysql 的正常模式，还需要 *intercept* 返回 greet 数据包信息。

4） 为什么 ab 发出的请求，在压力很大的场合，TCPCopy 丢失率非常高

答：TCPCopy 是基于 session 的，一个连接一个 session，由 4 元组构成，源 IP 地址，源端口，目的 IP，目的端口。在同一台机器 ab 发出的请求是基于短连接的，可变的一般只有源端口号，再加上压力大时候，QPS 往往非常高，ab 重复利用源端口的速度非常快，导致测试服务器（测试服务器比在线服务器至少多一个跳转）还没有来得及处理具有同一四元组的前一个 session 的数据，后面的 session 就过来了。这样的问题比较类似于 TCP 的 timewait 问题，因此我们可以采用如下策略：

a）从多台机器（最好同一网段）发送 ab 请求到测试服务器，这样可变的不仅仅是端口号，还有 IP 地址

b）采用长连接的压力测试工具，比如 *httpsender*

5）跨网段机器之间无法复制请求

答：请确认跨网段的测试服务器能不能抓到在线的请求数据包，如果能，请参考 QA 19；如果不能，采用如下方法进行诊断。

请先在同一网段复制请求，看能不能成功。如果同一网段可以成功复制请求，那说明跨网段转发过程中遇到安全方面的问题（TCPCopy 一般是支持不同网段的请求复制的，由于不可控因素的存在，导致跨网段无法复制请求的情况比较常见），导致复制的数据包在中途被丢弃掉了。一般解决方案如下：

a） 如果条件允许，只在同网段进行流量复制

b） 先复制请求到同一网段，然后设置代理，把流量导向不同网段的服务器

c） 通过在 *tcpcopy* 端设置-c 参数，改变请求数据包的源 IP 地址成合法的 IP 地址（注意这个方案只能在 0.9.0 版本及其以后版本才有效果）

6）TCPCopy 崩溃了,怎么办？

答：感谢你发现进程崩溃的情况，请设置 *ulimit -c unlimited*，并在 configure.ac 文件中修改 CFLAGS（把-O2 改成-g），再运行一遍，这时候如果崩溃，会产生 core 文件，然后 *gdb ./tcpcopy core.xxx*，进入 coredump 文件，执行 bt，打印出堆栈情况，请把此堆栈情况通过电子邮件（wangbin579@gmail.com）发送过来。

7）发现复制过去的请求比在线多，这是怎么回事？

答：如果在线服务器前面有 LVS，这很正常，如果没有 LVS，对比在线 access.log 和测试服务器的 access.log，应该就能看出为什么多出请求了。

具体可以参考例子分析：https://github.com/wangbin579/tcpcopy/issues/113

8）我下载了 TCPCopy，但我想使用 MysqlCopy，怎么编译？

答：从 github 上面下载的最新版本，在 *configure* 的时候增加相应模式，比如：

--enable-mysqlsgt	mysql skip-grant-tables mode
--enable-mysql	mysql mode

9）能不能只复制 post 请求？

答：目前 TCPCopy 还不能区分哪些请求是 post 请求，目前可以采用的策略就是先复制请求给 web 服务器代理，在 web 代理服务器上面抽取出 post 请求，再转发给待测试的服务器。

10）基于用户 session 的请求复制过去以后都被测试服务器给拒绝了，为什么？

答：一般方法有三个：

a） 进一步对 TCPCopy 进行二次开发,可参考 mysql 模式

b） 在测试服务器的上层应用去掉校验过程，类似 mysql 采用 skip 方式跳过校验过程

c） 先把请求复制给标准的 web 服务器，在 web 服务器端进行应用层面的修改，然后再转发给后端的测试服务器。

11）MysqlCopy 复制后，没有看到一个有效请求过来？

答：由于 mysql 协议是有状态的，中途截获请求，由于无法自动填补前面的内容，导致截获的请求是没有意义的，所以 MysqlCopy 需要能够捕获 mysql 的登入过程才可

以。

一般解决方案如下几种：

a）定期杀 TCP 连接

b）重启访问 mysql 的在线应用服务器

c）重启在线 mysql

12）错误日志文件在哪儿？

答：当你在/home/xxx/下运行/usr/local/bin/tcpcopy，那么 log 文件默认情况下就在/home/xxx/目录下，你也可以在执行 *tcpcopy* 或者 *intercept* 的时候设置-l 参数来指定错误日志文件的路径。

13）如何复制多个端口的数据到测试服务器中去？

答：例如复制 80 和 8080 端口的请求到测试服务器中去，方法如下（传统架构）：

./tcpcopy -x 80-192.168.0.2:80,8080-192.168.0.2:8080

14）如何放大在线压力？

答：可以通过多种方法，一般推荐-n 参数或者-f 参数。

15）如何缩小在线压力？

答：查看 *tcpcopy -h* ,看看有没有-r 参数，如果有，那说明此版本支持复制部分流量到测试服务器中去。

16）如何以 daemon 运行 *tcpcopy* 和 *intercept*?

答：加-d 参数即可。

17）如何在测试过程中访问测试服务器上面的测试应用

答：针对传统架构，一般可以利用 *intercept* 的-x 参数设置可通过 IP 地址列表（这些 IP 发出的请求一定要直接访问测试服务器，而不能再访问在线服务器上面的应用）或者参考老外的例子（http://globaldev.co.uk/2013/01/migrating-memcached/）

针对新架构，设置相应的路由即可。

18）由于安全原因，我们不想对外暴露 36524 端口，怎么办？

答：设置-b 参数，如设置为内网 IP 地址，那么只有目的地址为内网 IP 地址才能访问测试服务器的 36524 端口。

19）请求数据包都过来了，但服务器还是没有访问记录？

答：利用 *tcpdump* 抓包，数据包都过来了，说明已经到达了测试服务器的数据链路层。如果利用 *netstat* 查看是否有应用的连接，如果没有，说明数据包在 IP 层给 drop 掉了，请查看是否有 rpfilter 设置，如果有，去掉此设置，一般即可解决此问题。

如果没有设置 rpfilter，那么确认一下 *iptables* 设置有没有冲突，如果有，请先整理好 *iptables* 相关规则。

20）源 IP 地址为 127.0.0.1（localhost）的请求能转发到其它服务器吗？

答：如果直接复制过去，一般是不可行的，但通过-c 参数，改变 127.0.0.1 地址为本机器的 IP 地址，一般就能够复制到其它机器。

21）请求过去了，但实际在线请求丢失率还是很高，怎么办？

答：请首先确认测试机器的相关系统参数是否配置合理，这里举传统架构的几个常见例子：

a) 如果采用 IP Queue 的话，看看 IP Queue 模块本身是否丢包（cat /proc/net/ip_queue，具体详细内容参考 2.3.6），这种情况常见于压力比较大的场合

b) 由于测试服务器 ip_conntrack 的干扰，导致请求数据包大量丢失，详细内容参考 2.3.4 中的第 5 部分。

c) 新架构模式下，pcap 抓请求数据包，导致 *tcpcopy* 大量报“unsend:too many packets”，这时候换成 raw socket 方式来抓请求数据包或者采用 pfring pcap 来抓请求数据包，一般即可解决此问题。

d) 由于应用程序的文件句柄默认设置太小，导致在流量大的情况下应用程序的吞吐量始终上不去

这时候为了排除系统参数设置不合理的因素，你可以复制在线流量的一部分流量到测试系统（比如复制十分之一的流量到测试系统，测试系统的 QPS 是否也是在线的十分之一左右），看看请求丢失率是否还是很高，如果压力小的时候，请求丢失率不高，那一般就是系统参数设置不合理，如果请求丢失率还是很高，请以 debug 模式运行 TCPCopy 1 分钟，把错误日志文件（如果文件比较大，先把文件 zip 压缩一下或者 tar.gz 压缩一下）通过电子邮件（wangbin579@gmail.com）发送给我，最好还能附加上其它有用的信息，原则上信息越多越好。

22）市面上已经有了 *tcpdump* 和 Tcpreplay，为什么重复造轮子？

答：*tcpdump* 是用来抓包的，Tcpreplay 是用来重放的，仅仅在 TCP 层以下进行重放，连 TCP 协议都未能通过，所以重放这个说法不准确，容易误人子弟。TCPCopy 重放可以做到实时重放，也可以离线重放，这是针对大部分协议的重放，不仅仅只在 TCP

以下。

23）如何测试防火墙等网络设备？

答：把在线请求复制给代理，代理再把流量导向目的地，防火墙后面需要配置相应的应用，因为 TCPCopy 是针对 server 的流量复制。

24）TCPCopy 采用什么开源协议？

答：BSD 协议。

25）对线上的应用，会有影响吗？

答：只要测试系统独立于在线系统，业务上就不会有影响，另外，在消耗系统资源方面（带宽，内存，cpu），一般影响几乎可以忽略不计。

26）如何离线回放？

答：*configure* 的时候加上*--enable-offline* 参数（这种模式不能工作在实时复制）；离线回放还需要安装 pcap 库和 pcap 开发库（需要用到 pcap 库的头文件），如果系统还没有安装的话；另外运行的时候需要指定-i 参数。

举例（传统架构）：

./tcpcopy -x 110-xxx.xxx.xxx.148:110 -i ./110.pcap

这里 110.pcap（利用类似于 *tcpdump* 的工具来抓请求数据包，存放到 pcap 格式的文件中去）文件作为数据源，转发这里面的 pop3 请求到测试服务器 148 上面的 pop3 服务中去

27）传统架构下，如何充分利用 *intercept -x* 参数，达到多个架构层次的同时测试

答：假设在线是 nginx→memcached 应用，在线 nginx 有 10 台机器，memcached 有 3 台机器，测试服务器只有一台，测试服务器部署了整个应用(测试服务器上面的 nginx 配置是采用 127.0.0.1 访问本地 memcached 应用)。

测试的目的是复制在线的 nginx 两台机器到测试服务器，同时还想复制 memcached 机器上面的所有请求到测试服务器上去，如何做到这一点？

我们在测试服务器设置 iptables 如下：

iptables -I OUTPUT -p tcp --sport 11211 -j QUEUE

iptables -I OUTPUT -p tcp --sport 80 -j QUEUE

运行*./intercept -x 127.0.0.1 -d*

这样配置后，除了 127.0.0.1 发起的请求不会被 *intercept* 丢弃掉外，其它所有请求都会被 drop 掉。

在线服务器运行 TCPCopy client：

复制所有 memcached 流量，在所有 memcached 机器运行如下命令：

./tcpcopy -x 11211-测试服务器 IP 地址:11211 -d

复制两台在线 nginx 流量，在两台在线服务器运行如下命令：

./tcpcopy -x 80-测试服务器 IP 地址:80 -d

这样就达到了整个系统复杂的测试。

28）如果复制请求过去后，中途源 IP 地址被修改了，还能工作吗？

答：这种场合，TCPCopy 无法正常工作，原因如下：

比如复制内网请求到外网中去，由于抓的在线包的源 IP 地址是内网地址，发送的路由信息（*tcpcopy* 发送给 *intercept* 的路由信息）也是内网地址的信息，访问测试服务器的源 IP 地址是外网地址，这样 IP 地址就不一致，当 TCP 返回第二次握手信息给测试服务器 IP 层的时候，*intercept* 就会去找路由信息，由于测试服务器 TCP 协议返回的数据包的目的 IP 地址是外网地址，利用外网地址去找如何返回的信息，是找不到的，因此第二次握手数据包无法返回给 *tcpcopy*，从而导致会话无法继续下去。

如果大家遇到此问题，可以采用离线回放方式，就可以避开源 IP 地址修改的问题。

29）传统架构下，*intercept* 报 cannot set mode:: Connection refused

答：一般是因为 IP Queue 模块没有启动导致的,还有可能是 IP Queue 模块已经被破坏掉了。

30）错误日志文件报大量 Message too long，怎么办？

答：请使用 0.5 或者以上版本

31）出现大量 warn 级别的日志 *unsend:too many packets*，是什么意思？

答：原因可能有很多，候选原因如下：

a）测试服务器处理速度慢，导致测试服务器积累的数据包越来越多。

这种情况常见于传统压力测试工具发出的请求

b）测试服务器对某些请求无响应，导致缓存的数据包的数目迅速上升

c）传统架构下，系统参数设置不合理，比如没有设置好 IP Queue 系统参数，测试服务器传递给 *intercept* 的过程中就丢失了很多响应数据包，导致会话无法继续下去，缓存的数据包越积越多

d）新架构下 pcap 抓请求数据包的时候出现了大量丢包

这时候推荐采用 raw socket 方式来抓请求包（*configure --enable-advanced*）或者

pfring pcap 抓请求包

e）……

你可以尝试在压力小的时候，请求丢失率是否还是很高，如果压力小的时候丢失率还是很高，*./configure --enable-debug*，运行 1 分钟，把错误日志文件（压缩成 zip 格式或者 tar.gz 格式）发送给我，我来查明具体可能原因。

32）为什么离线回放不能实时转发流量？

答：考虑到性能方面，不同版本代码不一样，分开编译，性能会更高；考虑到可使用性，由于离线回放方式依赖于 libpcap 库，而在线实时方式默认不需要这个库，所以离线方式和在线方式分开编译。

33）利用 *tcpdump* 抓包，为什么推荐只抓请求包？

由于离线回放只需要在线的请求包，不需要在线的响应包，如果全抓会影响 *tcpcopy* 的处理速度，而且还浪费 pcap 文件所占的磁盘空间。

34）如何保证 github 下载的版本可用？

修改代码后，我们会进行回归测试，只有通过了回归测试，我们才会更新到 github master 版本。

35）TCPCopy 只能运行在 Linux 下面吗？

目前 TCPCopy 只能运行在 linux，未来会在多个操作系统下支持新架构（仅仅支持新架构，不会支持传统架构）。

36）TCPCopy 只能 root 运行吗？

答：需要 root 权限，比如 raw socket 和与内核打交道的函数需要 root 权限才能运行

37）传统架构下，*intercept* 能不能在同一台机器运行多份？

答：经我们测试，传统架构不行，而且对于 IP Queue 模块或者 NFQueue 模块也没有必要运行多份，设置多条 *iptables* 命令即可。

新架构可以在同一台机器运行多份。

38）先关闭 *tcpcopy*,还是先关闭 *intercept*?

答：一般推荐先关闭 *tcpcopy*，再关闭 *intercept*，但如果是分布式测试，且想快速关闭整个测试过程，那么推荐先关闭 *intercept*。

39）怎么加入到这个项目？

答：首先很欢迎加入这个项目，目前代码都在 github；其次任何修改，哪怕是英文注释修改，我们都是非常欢迎的。

目前参与人员，可以在下面地址看到：

https://github.com/wangbin579/tcpcopy/contributors

我们希望队伍越来越大，共同为中国开源作出一份贡献。

40）如何复制 pop3 请求？

答：跟一般复制差别不是很大，但还是需注意以下内容：

a）对于在线和测试服务器，pop3 用户名和密码必须一致

b）长连接场合下，请求复制效果可能不是很理想，因为针对此类型的应用，还没有类似于 mysql 的自动补充登入的过程。

41）离线回放，错误日志文件报大量 truncated packets warning，是怎么回事？

答：由于抓包过程中并没有把请求内容抓全，导致了此问题。

建议利用 *tcpdump* 抓包的时候，指定-s 大小为 65535 或者-s 0

42）传统架构下，TCPCopy client 需要 *IP Queue 模块*支持吗？

答：不需要

43）在线服务器报“*ip_conntrack: table full, dropping packet*”，这是怎么回事？

答：在在线服务器内，由于 ip_conntrack 是在 IP 层中做的，而 *tcpcopy* 转发数据包，默认利用 raw socket output 接口，因此会走部分 IP 函数，导致了 ip_conntrack 瞎推测连接状态。

为了解决此问题，可以采用如下四种方案（均在在线服务器内操作）：

a）去除 ip_conntrack 内核模块

modprobe -r ip_conntrack

b）设置对 *tcpcopy* 转发的包不进行 track

iptables -t raw -A OUTPUT -p tcp --dport 转发目的端口 -j NOTRACK

c）加大 ip_conntrack_max 值

比如：

echo "262144" > /proc/sys/net/ipv4/ip_conntrack_max

d) 采用 packet injection 技术从数据链路层发送数据包（--enable-dlinject）

44）传统架构下，*sudo iptables -I OUTPUT -p tcp --sport 36524 -j QUEUE*，有什么问题？

由于 *intercept* 默认采用 36524 端口进行监听，所以设置上述命令，会导致 *tcpcopy* 和 *intercept* 无法建立连接，从而无法继续工作下去。

如果应用是 36524 端口，那么你需要改变 *intercept* 监听的端口，可以采用-p 参数来改变监听端口。

45）离线回放，编译出现找不到 pcap.h 文件，怎么办？

需要安装 pcap 的 dev 库，比如 centos 机器上：

yum install -y libpcap-devl

46）为什么加了‘&’，*tcpcopy* 还经常退出？

仅仅‘&’是不够的，还需要加 nohup，不过建议采用加-d 参数的方式

47）传统架构下，每天运行到某一个时刻，tcpcopy 就无法把请求复制给测试系统了，为啥？

答：请利用“iptables -L”检测一下，看看相应的设置是否已经被定期清理掉了

48）遇到问题怎么办？

访问 https://github.com/wangbin579/tcpcopy/issues，看看有没有类似的问题。

如果没有，最好*./configure --enable-debug* 运行，输出 1 分钟的 log（如果压力大，可以在 *tcpcopy* 命令端加上-r 参数，复制部分在线流量到测试服务器中去;如果文件很大，先压缩成 zip 格式或者 tar.gz 格式），发送过来，当然再加上辅助信息更好（比如是否是虚拟机），原则是给的信息越多越好。

49）测试服务器报“*ip_conntrack: table full, dropping packet*”，这是怎么回事？

答：测试服务器报这个，其实已经跟 *tcpcopy* 没有关系了。

一般可以这样解决（均在测试服务器内操作）：

a）去除 ip_conntrack 内核模块

modprobe -r ip_conntrack

b）设置对请求包不进行 track

iptables -t raw -A PREROUTING -p tcp --dport 转发目的端口 -j NOTRACK

c）加大 ip_conntrack_max 值

比如：

echo "262144" > /proc/sys/net/ipv4/ip_conntrack_max

50）按照 README 文件来操作，命令都一样，为啥不见请求过来？

答：完全按照例子中的命令，一般是不行的，因为不同的应用有不同的端口，所以相应的命令也需要做相应的修改，请理解好 TCPCopy 原理后再试验。

51）遇到问题后，该发哪些信息？

答：俗话说，不幸的家庭各有个的不幸，仅仅告诉啥问题，一般是很难推测出原因的，所以希望大家尽可能地提供详细的信息，方便问题诊断。

52）如何查看请求复制的效果？

答：一般可以通过这些值来分析出复制的效果：

a) 在线请求数据包方面：

syn cnt:xxx	抓到了多少在线 syn 数据包
all clt packs:xxx	抓到了多少在线数据包
clt cont:xxx	抓到了多少带有 payload 的在线数据包

b) 复制转发数据包方面：

send Packets:xxx	发送了多少数据包给测试服务器
send content packets:xxx	发送了多少带有 payload 的数据包给测试服务器

c) 响应方面：

conns:xxx	已经成功了建立了多少连接
resp packs:xxx	收到了多少带有 payload 的响应包
c-resp packs:xxx	收到了多少响应包

52）出现大量 send to:Inappropriate ioctl，怎么回事?

答：如果测试机器内核系统参数设置合理的话，一般是在线机器压力比较大的时候，才会出现这类问题，请设置在线机器的相关的系统参数或者加大-C 参数。

53）传统架构下，出现 Set netlink socket(5) mode failed 或者 can't set mode,check if ip queue is up:Illegal seek，怎么回事？

答：请查看 IP Queue 模块是否已经启动

54）传统架构下，*intercept* 报 nl recvfrom (No buffer space available)，怎么办？

答：加大设置 *intercept* 所在的机器的网络内核 buffer，比如：

```
sysctl -w net.core.rmem_max=16777216
```

sysctl -w net.core.wmem_max=16777216

如果还不行，采用新架构吧。

55）传统架构下，测试机器全都是 SYN_RECV 状态，怎么办？

答：*iptables* 针对端口的设置是否正确。如果没有问题，可能原因如下：

1）IP Queue 模块虽然加载了，但由于 NFQueue 的干扰，导致 IP Queue 模块工作不正常，这时候要么只用 NFQueue,要么卸载 NFQueue 相关内核模块，再重启 IP Queue 内核模块（具体参考：https://github.com/wangbin579/tcpcopy/issues/60）

2）中途设备只让第一次握手数据包过去，而不让后续的第三次握手数据包过去

3）*tcpcopy* 和 *intercept* 不兼容

比如 0.6.0 版本的 *tcpcopy* 和 0.8.0 版本的 *intercept* 是不兼容的

56）TCPCopy 能修改应用层的数据（比如 url 参数）吗？

答：一般需要二次开发，当然最简单的方式就是先用 TCPCopy 转发请求给应用服务器（比如 http 请求，可以采用 web 服务器），在应用服务器上面进行修改，再转发给测试服务器。

57）通过查看 *tcpcopy* 的日志，看到 c-resp packs 的数值不为 0，怎么上层应用服务器没有看到访问日志？

答：一般是在刚运行的时候，因为访问量比较小，操作系统还没有把访问日志输出到日志文件，所以还不能看到。

58）在同一网段，复制请求给虚拟机，在测试服务器抓不到任何复制过来的数据包，而在在线服务器能抓到，怎么回事？

答：这个应该是虚拟机把复制的包给丢弃掉了，在同一网段的虚拟机，相当于跨网段的情况。

解决方案：1）请复制给同一台机器上面的应用代理，再转发给其它机器 2）采用离线回放，避开数据包的安全检查 3）*tcpcopy* 运行的时候加上-c 参数，改变数据包的源 IP 地址

59）sh autogen.sh 运行错误，怎么办？

答：参考 https://github.com/wangbin579/tcpcopy/issues/63

60）复制在线请求，在测试服务器只能看到内网 IP 地址（或者可信任的 IP 地址）过来

的请求过来，怎么回事？

答：据统计，目前查询到的原因有如下两个：

1）可能你的机器没有外网 IP 地址（或者外网网卡），导致 IP 模块在流量进入的时候对外网 IP 地址的数据包进行了过滤。这时候给机器分配外网网卡和外网 IP 地址即可解决。

2）网卡 IP 地址设置错误，比如用 *ifconfig* 针对网卡设置了错误的 IP 地址

61）传统架构下除了在 *intercept* 设置-x 参数外，还有哪些手段能够达到访问测试服务器的目的？

答：如果用户熟悉 iptables 设置，可以通过 iptables 设置命令达到类似的效果

举例如下：

http://globaldev.co.uk/2013/01/migrating-memcached/

62）如何利用新架构？

答：TCPCopy 新架构使用方法：

http://blog.csdn.net/wangbin579/article/details/8950282

TCPCopy 新架构具体复杂应用实例：

http://blog.csdn.net/wangbin579/article/details/8994601

利用 pcap injection 从数据链路层发包（0.9.0 版本及其以上才支持）：

http://blog.csdn.net/wangbin579/article/details/10148247

63）复制在线的请求到 127.0.0.1 上面的应用，上层应用没有显示任何 log？

答：参考 https://github.com/wangbin579/tcpcopy/issues/115

64）复制内外网请求到没有外网网卡的测试服务器，结果只看到内网请求复制过来了？

答：这个问题类似于问题 60。

外网请求复制给测试服务器后，由于测试服务器只有内网网卡，可能 IP 模块认为这些请求即使被处理了，也回不去，就把这些请求给丢弃了。

解决方案：

1）加外网网卡,分配外网 IP 地址。

2）通过 *tcpcopy* 的-c 参数，改变外网请求的源 IP 地址

65）如果对海量请求进行复制？

答：请采用新架构，并且采用 pcap 模式（最好有 pfring 配合），利用 pcap filter 来

分割请求的捕获和响应包的捕获，这样就能够实现多个 *tcpcopy* 和多个 *intercept* 实例共同完成请求的复制和响应包的返回，从而达到超级压力下的海量测试。

采用 pcap injection 技术，还可以进一步提升 tcpcopy 的性能，具体参考：

http://blog.csdn.net/wangbin579/article/details/10148247

66）如何从数据链路层发包？

答：参考 http://blog.csdn.net/wangbin579/article/details/10148247

100）Why not just go through normal tcp stack to maintain connections between production server and test server ? That would simplify the code dramatically, would scarifies some performance though.

Answer：

If it goes through normal tcp stack to maintain connections between production server and test server, external requests would be transferred to internal requests. The characteristics of external network, such as latency, would be lost. Moreover, it will affect the online machines, as using normal tcp stack would occupy system resources, such as ports.

9 总结

TCPCopy 的使命有如下几个方面：

a) 让每个潜在的 IT 程序员能够成为一名优秀的架构师，不再仅仅依赖经验做事情
b) 尽可能使上线过程轻松，bug free
c) 为开源做贡献
d) 改变上线前测试的手段

总体来说，如果你对上线没有信心，如果你的单元测试不够充分，如果你对新系统不够有把握，如果你对未来的请求压力无法预测，如果你想对比诸如 apache 和 nginx 的性能，如果你想放大在线流量，如果你想 profile 测试应用，如果你想成为出色的架构师， TCPCopy 可以帮助你解决上述难题。