TCPCopy Manual

NetEase, Inc.

2013.04.07

Table of Contents

1 Overview

TCPCopy is an online TCP duplication tool mainly developed by NetEase and also contributed by developers in other companies (e.g., Taobao) and has been widely used in companies in China (e.g., NetEase, Taobao, Sohu). It could copy the TCP live flow on the online server to a target server, and thus bring the complexity of online environments to the target server. It is especially useful for live testing and reducing errors before a system being deployed online. For example, when you want to migrate from Apache to Nginx, TCPCopy can help you test it. While Apache is running online, TCPCopy can copy the TCP flows from Apache to Nginx. To Nginx, the TCP flows are just forwarding to it. This will not affect Apache at all except cost a little network bandwidth and CPU load.

TCPCopy could be used in the following areas.

1) Performance testing, such as finding performance bottlenecks and comparing performances of different programs.

2) Smoke testing

3) Regression testing

4) Live testing

5) …

TCPCopy includes two versions: online version and offline version. They differ in how they obtain packets. While the online version captures the packets from the online server in real-time, the offline version reads packets from the pcap file which contains packets captured previously. As the offline version relies on packet capturing tools which may lose packets and impact the online file IO and it is also difficult to accommodate timing issues in replay, we don't suggest using the offline version under high pressure. In the following, we refer TCPCopy to the online version by default.

2 How Does TCPCopy Work?

Generally speaking, TCPCopy copies packets on the online server, modifies TCP/IP headers, and sends modified packets to the target server. In this way, TCP applications on the target server will consider the packets from the online server as online requests from end-users. As most applications on the internet are based on TCP protocol, TCPCopy could be widely used for applications based on HTTP, memcached, POP3, etc.

The framework of TCPCopy can be seen in Figure 1.

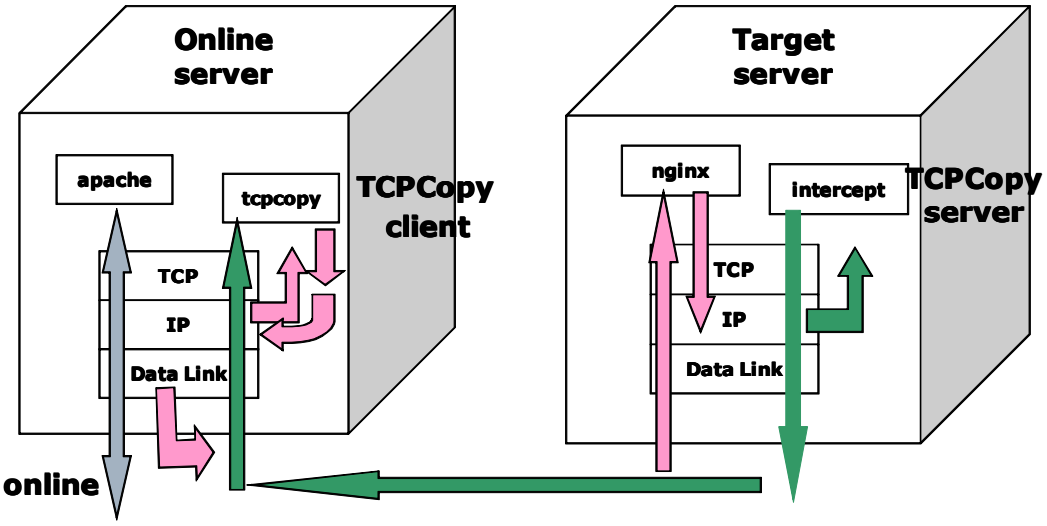

Figure 1. Framework of TCPCopy

As shown in Figure 1, TCPCopy consists of two parts: the TCPCopy client (***tcpcopy***) and the TCPCopy server (***intercept***). While the TCPCopy client runs on the online server and captures the online requests, TCPCopy server runs on the target server and does some assistant work, such as passing response info to the TCPCopy client and filtering outbound traffic.

The TCPCopy client (*tcpcopy*) utilizes *raw socket input* technique to capture the online packets in the IP layer and modify their attributes (e.g. acknowledgment number, destination IP and destination port), and uses *raw socket output* technique to send packets to the target server.

The TCPCopy server (*intercept*) is responsible for passing the response header to the TCPCopy client. By setting the *iptables* command, locally generated response packets will be sent to the corresponding kernel module (*ip_queue* or *nfqueue*), and then the kernel module will attempt to deliver the packets to *intercept*, which will extract response header information and determine whether to drop the packet or not. To make the TCPCopy client send the next packet, the TCPCopy server often needs to send the response header to the TCPCopy client using a TCP connection. When the TCPCopy client receives the response header, it utilizes the header information to modify the attributes of online packets and continues to send another packet.

It should be noticed that the responses from the target server are dropped in the IP layer of the target server and not return to the end-user by default. There are two reasons to do this.

1) It will not burden the output bandwidth, and can save money.

2) It will not influence the TCP/IP protocol of the end-user.

3 TCPCopy vs Tcpreplay

Tcpreplay is a suite of utilities for UNIX systems for editing and replaying network traffic which was previously captured by tools like *tcpdump* and *ethereal/ wireshark*. The goal of tcpreplay is to provide reliable and repeatable means for testing a variety of network devices such as switches, router, firewalls, network intrusion detection and prevention systems (IDS and IPS).

TCPCopy and tcpreplay are different in the following aspects. First, TCPCopy can be used by high-level applications, while tcpreplay can not. Since tcpreplay is unable to synchronize Seq/Ack numbers or maintain any TCP state, it is unable to establish a valid TCP session with a server. Thus, it is ineffective for high-level applications. However, TCPCopy can capture the online flow, send the captured data to a target server in real-time and establish valid TCP sessions with the target server, so it can be used by high-level applications.

Second, compared with tcpreplay, TCPCopy has less influence on the online server. Tcpreplay often works with packets captured and saved by sniffers/ packet capturing tools, such as *tcpdump*. When the online system is busy, *tcpdump* may influence the online file IO and the packet loss rate may be high. Moreover, as the captured packets should be saved, *tcpdump* seldom captures packets continuously for a few days. On the contrary, TCPCopy does not need many file IO operations, and can copy online packets to a target server in real-time for several days.

Third, TCPCopy keeps more online features than tcpreplay. When replaying with tcpreplay, the source IP address in the packet may be modified and different from online environments. Also it is hard for tcpreplay to keep the packets' online delay feature precisely because of the timing issues, which makes the replaying effect different from online environments. By contrast, TCPCopy captures and sends packets in real-time, so the packets sent to the target server may keep many characteristics of online packets, such as delay and client packet attributes.

Lastly, it is more complex to use tcpreplay than TCPCopy. Compared with tcpreplay, we think TCPCopy is easier to learn and easier to use.

In summary, TCPCopy could be used to test your applications on TCP layer or above effectively with good performance.

4 How to Use TCPCopy?

4.1 Getting TCPCopy Installed

To use TCPCopy in your work, you first have to install it.

To install TCPCopy:

1) **$** *git clone http://github.com/wangbin579/tcpcopy*

2) **$** *cd tcpcopy*

2) **$** *sh autogen.sh*

3) **$** *./configure*

4) **$** *make*

5) **#** *make install*

Advanced Options: There are quite a few configuration options for TCPCopy which allow you to control a lot of things. They are as follows.

--enable-debug	compile TCPCopy with debug support (saved in a log file)
--enable-mysqlsgt	run TCPCopy at mysql skip-grant-tables mode
--enable-mysql	run TCPCopy at mysql mode
--enable-offline	run TCPCopy at offline mode
--enable-pcap	run TCPCopy at pcap mode
--enable-thread	run the TCPCopy server (*intercept*) at multi-threading mode
--enable-nfqueue	run the TCPCopy server (*intercept*) at *nfqueue* mode

4.2 Launching TCPCopy on Source and Target Servers

After installing TCPCopy, you have to deploy the TCPCopy client (*tcpcopy*) on the online source server and the TCPCopy server (*intercept*) on the target server.

On the target server (root privilege is required):

Using *ip queue*（kernel < 3.5）:

1) *# modprobe ip_queue # if not running*

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

3) *# ./intercept*

Or

Using *nfqueue* (kernel >=3.5):

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

2) *# ./intercept*

On the online source server (root privilege is required):

 # ./tcpcopy -x localServerPort-targetServerIP:targetServerPort

4.3 Parameters for the TCPCopy Server (*intercept*)

On the target server, you could use "*intercept -h*" to get the following help information.

-x <passlist,>	passed ip list through firewall Format: ip_addr 1, ip_addr 2 …
-p <num>	set the TCP port number to listen on. The default number is 36524.
-s <num>	set the hash table size for intercept. The default value is 65536.
-l <file>	save log information in <file>
-t <num>	set the router item timeout limit. The default value is 120 sec. It should be set larger when connections are idle longer than the default value.
-P <file>	save PID in <file>, only used with -d option
-b < ip_addr >	interface to listen on (default: INADDR_ANY, all addresses)
-v	intercept version
-h	print this help and exit
-d	run as a daemon

4.4 Parameters for the TCPCopy Client (*tcpcopy*)

On the online server, you could use "*tcpcopy -h*" to get the following help information.

-x <transfer,>	use <transfer,> to specify the IPs and ports of the source and target servers. Suppose 'sourceIP' and 'sourcePort' are the IP and port number of the source server you want to copy from, 'targetIP' and 'targetPort' are the IP and port number of the target server you want to send requests to, the format of <transfer,> could be as follows: 'sourceIP:sourcePort-targetIP:targetPort,...'. Most of the time, sourceIP could be omitted and thus <transfer,> could also be: 'sourcePort-targetIP:targetPort,...'. As seen, the IP address and the port number are segmented by ':' (colon), the sourcePort and the targetIP are segmented by '-', and two 'transfer's are segmented by ',' (comma). For example, './tcpcopy -x 80-192.168.0.2:18080' would copy requests from TCP port '80' on current server to the target port '18080' of the target IP '192.168.0.2'.
-c < ip_addr >	change the localhost client IP to this IP address when sending to the target server. For example, './tcpcopy -x 8080-192.168.0.2:8080 -c 192.168.0.1' would copy

	requests from port '8080' of current online server to the target port '8080' of target server '192.168.0.2' and modify the client IP to be '192.168.0.1' when client IP is localhost.
-i <file>	set the pcap file used for tcpcopy to <file> (only valid when TCPCopy is configured to run at enable-offline mode)
-n <num>	use <num> to set the replication times when you want to get a copied data stream that is several times as large as the online data. The maximum value allowed is 1023. As multiple copying is based on port number modification, the ports may conflict with each other, in particular in intranet applications where there are few source IPs and most connections are short. Thus, tcpcopy would perform better when less copies are specified. For example, './tcpcopy -x 80-192.168.0.2:8080 -n 3' would copy data flows from port '80' on the current server, generate data stream that is three times as large as the source data, and send these requests to the target port '8080' on '192.168.0.2'.
-f <num>	use this parameter to control the port number modification process and reduce port conflictions when multiple tcpcopy instances are running. The value of <num> should be different for different tcpcopy instances. The maximum value allowed is 1023.
-m <num>	set the maximum memory allowed to use for tcpcopy in megabytes, to prevent tcpcopy occupying too much memory and influencing the online system. When the memory exceeds this limit, tcpcopy would quit automatically. The parameter is effective only when the kernel version is 2.6.32 or above. The default value is 512.
-M <num>	MTU value sent to backend (default 1500)
-t <num>	set the session timeout limit. If tcpcopy does not receive response from the target server within the timeout limit, the session would be dropped by tcpcopy. When the response from the target server is slow or the application protocol is context based, the value should be set larger. The default value is 60 seconds
-l <file>	save the log information in <file>
-r <num>	set the percentage of sessions transferred (integer range:1~100)
-p <num>	set the target server listening port. The default value is 36524.
-P <file>	save PID in <file>, only used with -d option

-h	print this help and exit
-v	version
-d	run as a daemon

4.5 Distributed Deployment of TCPCopy

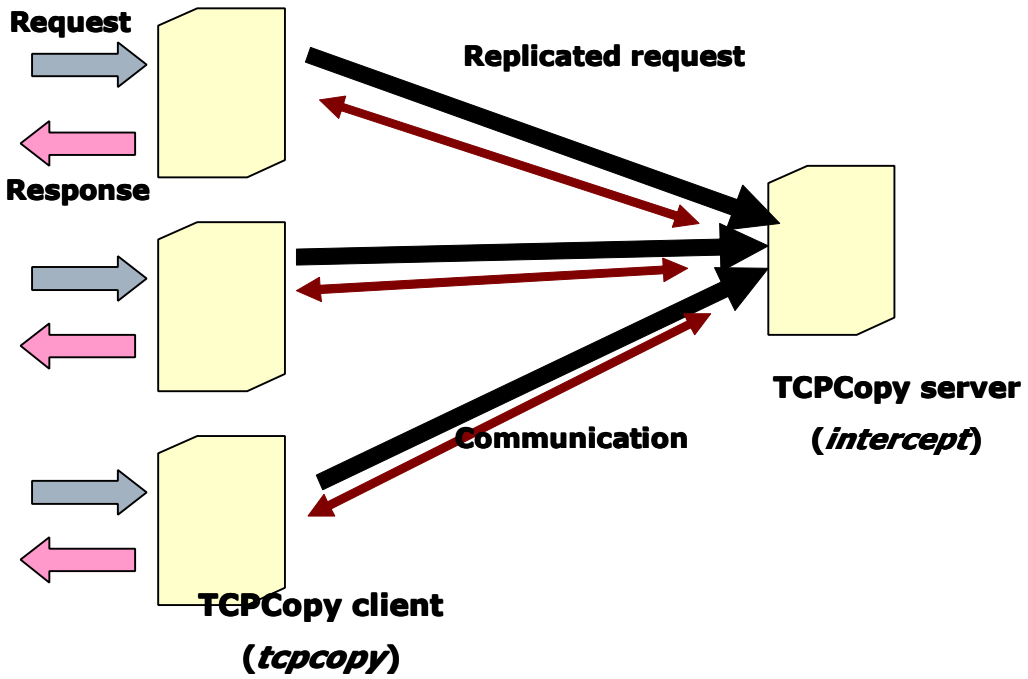

Figure 2. Distributed deployment of TCPCopy

With TCPCopy, we could test whether a server could sustain distributed attacking by copying online flow from multiple servers to a target server. Figure 2 shows an example of deploying TCPCopy in a distributed architecture.

4.6 An Example

Suppose there are two online servers, 1.2.3.25 and 1.2.3.26. And 1.2.3.161 is the target server. Port 11311 is used as the online source server port and port 11511 is used as the target server port. We use TCPCopy to test if 1.2.3.161 can process 2X requests than an online server can serve.

Here we use TCPCopy to perform the above test task.

1) download and install TCPCopy

2) on the target server (1.2.3.161, kernel 2.6.18)

 # modprobe ip_queue
 # iptables -I OUTPUT -p tcp --sport 11511 -j QUEUE
 # ./intercept

3) on the source online server (1.2.3.25)

 # ./tcpcopy -x 11311-1.2.3.161:11511

4) on the source online server (1.2.3.26)

 # ./tcpcopy -x 11311-1.2.3.161:11511

CPU load and memory usage is as follows:

1.2.3.25:

```
21158 appuser   15   0  271m 226m  756 S 24.2  0.9   16410:57 asyn_server
9168  root      15   0 18436  12m  380 S  8.9  0.1   40:59.15 tcpcopy
```

1.2.3.26:

```
16708 appuser   15   0  268m 225m  756 S 25.8  0.9   17066:19 asyn_server
11662 root      15   0 17048  10m  372 S  9.3  0.0   53:51.49 tcpcopy
```

1.2.3.1161:

```
27954 root      15   0  284m  57m  828 S 58.6  1.4 409:18.94 asyn_server
1476  root      15   0 14784  11m  308 S  7.7  0.3   49:36.93 intercept
```

Access log analysis:

1.2.3.25:

```
$ wc -l access_1109_09.log
  7867867,  2185 reqs/sec
```

1.2.3.26:

```
$ wc -l access_1109_09.log
  7843259,  2178 reqs/sec
```

1.2.3.161:

```
$ wc -l access_1109_09.log
  15705229, 4362 reqs/sec
```

request loss ratio:

(7867867 + 7843259 - 15705229) / (7867867 + 7843259) = 0.0375%

Clearly, the target server can process 2X of requests a source server can serve.

How is the CPU load? Well, the TCPCopy client (*tcpcopy*) on online server 1.2.3.25 used 8.9%, server 1.2.3.26 used 9.3%, while the TCPCopy server (*intercept*) on the target server consumed about 7.7%. We can see that the CPU load is low here, and so is the memory usage.

5 Influential Factors

There are several factors that could influence TCPCopy, which will be introduced in detail in the following sections.

5.1 *Raw Socket Input* Interface

TCPCopy utilizes *raw socket input* interface to capture packets on the online server. The system kernel may lose some packets when the system is busy. Thus, the related system parameters should be set appropriately.

5.2 *Raw Socket Output* Interface

TCPCopy utilizes *raw socket output* interface to send packets to a target server. The system kernel may encounter problems and not send all the packets successfully. For example, when the packet size is larger than MTU, *raw socket output* interface would refuse to send these large packets. In TCPCopy 0.5 or above versions, with our special processing, large packets are supported.

5.3 On the Way to the Target Server

When a packet is sent by the TCPCopy client (*tcpcopy*), it may encounter many challenges before reaching the target server. As the source IP address in the packet is still the end-user's IP address other than the online server's, some security devices may take it for an invalid or forged packet and drop it. In this case, when you use *tcpdump* to capture packets on the target server, no packets from the expected end-users will be captured. To know whether you are under such circumstances, you can choose a target server in the same network segment to do a test. If packets could be sent to the target server successfully in the same network segment but unsuccessfully across network segments, your packets may be dropped halfway.

To solve this problem, we suggest deploying the TCPCopy client (*tcpcopy*) and the TCPCopy server (*intercept*) on servers in the same network segment. There's also another solution with the help of a proxy in the same network segment. The TCPCopy client could send packets to the proxy and then the proxy would send the corresponding requests to the target server in another network segment.

5.4 OS of the Target Server

The target server may set *rpfilter*, which would check whether the source IP address in the packet is forged. If yes, the packet will be dropped in the IP layer.

If the target server could not receive any requests although packets can be captured by *tcpdump* on the target server, you should check if you have any corresponding *rpfilter* settings. If set, you have to remove the related settings to let the packets pass through the IP layer.

5.5 Applications on the Target Server

It is likely that the application on the target server could not process all the requests in time. On the one hand, bugs in the application may make the request not be responded for a long time. On the other hand, some protocols above TCP layer may only process the first request in the socket buffer and leave the remaining requests in the socket buffer unprocessed.

5.6 *Netlink Socket* Interface[1]

Packet loss also occurs when *ip queue* module transfers the response packet to the TCPCopy client (*tcpcopy*) under a high-pressure situation. By using command "cat /proc/net/ip_queue", you can check the state of *ip queue*.

If the value of *queue dropped* increases continually, ip_queue_maxlen should be set larger. For example, the following command modifies the default queue length 1024 to 4096.

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

If the value of *netlink dropped* increases continually, rmem_max and wmem_max should be set larger. Here is an example.

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

[1] intercept uses *ip queue* mode by default for kernel < 3.5

6 Release History

2011.09 version 0.1, TCPCopy released

2011.11 version 0.2, fix some bugs

2011.12 version 0.3, support mysql copy

2012.04 version 0.3.5, add support for multiple copies of the source request

2012.05 version 0.4, fix some bugs

2012.07 version 0.5, support large packets (>MTU)

2012.08 version 0.6, support offline replaying from pcap files to the target server

2012.10 version 0.6.1, support intercept at multi-threading mode

2012.11 version 0.6.3, fix fast retransmitting problem

2012.11 version 0.6.5, support *nfqueue*

2013.03 version 0.7.0, support lvs

7 Need Help?

Check error.log if you encounter some problems and feel free to report it to us (wangbin579@gmail.com).