

Tcpreplay 3.x Manual

Aaron Turner
<http://tcpreplay.synfin.net/>

27th February 2005

Contents

I	Before You Start	5
1	Overview	5
1.1	Corrections and additions to the manual	5
1.2	Getting Help	5
1.3	What tools come with tcpreplay?	5
1.4	What tools no longer come with tcpreplay?	5
2	Getting Tcpreplay working on your system	5
2.1	How can I get tcpreplay's source?	5
2.2	OS X and Debian users	5
2.3	What requirements does tcpreplay have?	6
2.4	How do I compile tcpreplay?	6
II	Basics	6
3	Basic Tcpreplay Usage	6
3.1	Replaying the traffic	6
3.2	Replaying at different speeds	6
3.3	Replaying the same file over and over again	7
3.4	Using Configuration Files	7
III	Advanced Usage	7
4	Output: Interfaces, Packets & Files	8
4.1	Replaying on multiple interfaces	8
4.2	Selectively sending or dropping packets	8
4.3	Replaying only a few packets	9
4.4	Skipping the first bytes in a pcap file	9
4.5	Replaying packets which are bigger then the MTU	9
4.6	Writing packets to a file	9
4.7	Extracting Application Data (Layer 7)	10
4.8	Replaying Live Traffic	10
4.9	Replaying Packet Capture Formats Other Than Libpcap	10
4.10	Replaying Client Traffic to a Server	10
4.11	Decoding Packets	10

5	Packet Editing	11
5.1	Rewriting MAC addresses	11
5.2	Randomizing IP addresses	11
5.3	Replaying (de)truncated packets	11
5.4	Rewriting Layer 2 with -2	11
5.5	Rewriting DLT_LINUX_SLL (Linux Cooked Socket) captures	12
5.6	Rewriting IP Addresses (pseudo-NAT)	12
5.7	Advanced pseudo-NAT	12
5.8	IP Endpoints	12
5.9	Unifying Dual-Outputs	13
6	Tcpprep Usage	13
6.1	What is tcpprep?	13
6.2	What are these 'modes' tcpprep has?	13
6.3	Splitting traffic based upon IP address	13
6.4	Auto Mode	14
6.4.1	How does Auto/Bridge mode work?	14
6.4.2	How does Auto/Router mode work?	14
6.4.3	Determining Clients and Servers	14
6.4.4	Client/Server ratio	14
6.5	Selectively sending/dropping packets	14
6.6	Using tcpprep cache files with tcpreplay	14
6.7	Commenting tcpprep cache files	14
7	Flowreplay Usage	15
7.1	How flowreplay works	15
7.2	Running flowreplay	15
8	Tuning OS's for high performance	15
8.1	Linux 2.4.x	15
8.2	*BSD	16
9	Understanding Common Error and Warning Messages	16
9.1	Can't open eth0: libnet_select_device(): Can't find interface eth0	16
9.2	Can't open lo: libnet_select_device(): Can't find interface lo	16
9.3	Can't open eth0: UID != 0	16
9.4	100000 write attempts failed from full buffers and were repeated	16
9.5	Invalid mac address: 00:00:00:00:00:00	16
9.6	Unable to process test.cache: cache file version mismatch	16
9.7	Skipping SLL loopback packet.	16
9.8	Packet length (8892) is greater then MTU; skipping packet.	17
9.9	Why is tcpreplay not sending all the packets?	17

10 Required Libraries and Tools	17
10.1 Libpcap	17
10.2 Libnet	17
10.3 Libpcapnav	17
10.4 Tcpdump	18
IV Other Resources	18
11 Other pcap tools available	18
11.1 Tools to capture network traffic or decode pcap files	18
11.2 Tools to edit pcap files	18
11.3 Other useful tools	18

Part I

Before You Start

1 Overview

1.1 Corrections and additions to the manual

I've tried to keep this document up to date with the changes in tcpreplay, but occasionally I get too busy, make a mistake or just forget something. If you find anything in this document which could be improved upon, please let me know.

1.2 Getting Help

If you still have a question after reading the Tcpreplay manual and FAQ, please contact the Tcpreplay-Users <tcpreplay-users@lists.sourceforge.net> mailing list. Note that if you ask a question which has clearly been covered in either the manual or FAQ, you will most likely be told to RTFM.

1.3 What tools come with tcpreplay?

- tcpreplay - replay packets stored in a pcap file as they were captured
- tcprewrite - edit packets stored in a pcap file
- tcpprep - a pcap pre-processor for tcpreplay/tcprewrite
- flowreplay¹ - connects to a server(s) and replays the client side of the connection stored in a pcap file

1.4 What tools no longer come with tcpreplay?

Recently, other people and projects have developed better versions of two applications that ship with tcpreplay 2.x:

- pcapmerge - merges two or more pcap files into one. Ethereal now ships with a more powerful application called 'mergcap'.
- capinfo - displays basic information about a pcap file. Ethereal now ships with a more powerful application of the same name.

2 Getting Tcpreplay working on your system

2.1 How can I get tcpreplay's source?

The source code is available in tarball format on the tcpreplay homepage: <http://tcpreplay.sourceforge.net/> I also encourage users familiar with Subversion to try checking out the latest code as it often has additional features and bugfixes not found in the tarballs.

svn checkout <https://www.synfin.net:444/svn/tcpreplay/trunk> tcpreplay-3

2.2 OS X and Debian users

There are OS X and Debian packages available via Fink and APT respectively. Both packages are maintained by other people and occasionally are out of date with the latest source code release.

¹Flowreplay is still "alpha" quality and is not usable for most situations. Anyone interested in helping me develop flowreplay is encouraged to contact me.

2.3 What requirements does tcpreplay have?

1. Libnet² 1.1.x or better (1.1.3 fixes a checksum bug)
2. Libpcap³ 0.6.x or better (0.8.3 or better recommended)
3. To support the packet decoding feature you'll need tcpdump⁴ installed.
4. You'll also need a compatible operating system. Basically, any *NIX operating system should work. Linux, *BSD, Solaris, OS X and others should all work. If you find any compatibility issues with any *NIX OS, please let me know.

2.4 How do I compile tcpreplay?

Two easy steps:

1. As a normal user: `./configure && make`
2. As root: `make install`

There are some optional arguments which can be passed to the 'configure' script which may help in cases where your libnet, libpcap or tcpdump installation is not standard or if it can't determine the correct network interface card to use for testing. If you find that configure isn't completing correctly, run: `./configure -help` for more information.

You may also choose to run 'make test -i' as root:

- make test is just a series of sanity checks which try to find serious bugs (crashes) in tcpprep and tcpreplay.
- make test requires at least one properly configured network interface. If the configure script can't guess what a valid interface is you can specify it with the `-with-testnic` and `-with-testnic2` arguments.
- If make test fails, often you can find details in `test/test.log`.
- OpenBSD's make has a bug where it ignores the MAKEFLAGS variable in the Makefile, hence you'll probably want to run: `make -is test` instead.

Part II

Basics

3 Basic Tcpreplay Usage

3.1 Replaying the traffic

To replay a given pcap as it was captured all you need to do is specify the pcap file and the interface to send the traffic out of:

```
tcpreplay -i eth0 sample.pcap
```

3.2 Replaying at different speeds

You can also replay the traffic at different speeds than it was originally captured⁵. To support this, tcpreplay supports four different flags: `-R`, `-r`, `-m`, and `-p`

Some examples:

²<http://www.packetfactory.net/libnet/>

³<http://www.tcpdump.org/>

⁴<http://www.tcpdump.org/>

⁵Tcpreplay makes a "best" effort to replay traffic at the given rate, but due to limitations in hardware or the pcap file itself, it may not be possible. Capture files with only a few packets in them are especially susceptible to this.

- To replay traffic as fast as possible:
tcpreplay -R -i eth0 sample.pcap
- To replay traffic at 10Mbps:
tcpreplay -r 10.0 -i eth0 sample.pcap
- To replay traffic 7.3 times as fast as it was captured:
tcpreplay -m 7.3 -i eth0 sample.pcap
- To replay traffic at half-speed:
tcpreplay -m 0.5 -i eth0 sample.pcap
- To replay at 25.5 packets per second:
tcpreplay -p 25.5 -i eth0 sample.pcap

3.3 Replaying the same file over and over again

Using the loop flag (-l) you can specify that a pcap file will be sent two or more times⁶:

- To replay the sample.pcap file 10 times:
tcpreplay -l 10 -i eth0 sample.pcap
- To replay the sample.pcap an infinitely or until CTRL-C is pressed:
tcpreplay -l 0 -i eth0 sample.pcap

3.4 Using Configuration Files

Tcpreplay offers the options of specifying configuration options in a config file in addition to the traditional command line. Each configuration option has an equivalent config file option which is listed in the tcpreplay man page. To specify the configuration file you'd like to use, use the -f <filename> option.

Configuration files have one option per line, and lines beginning with the pound sign (#) are considered comments and ignored. An example config file follows:

```
# send traffic out 'eth0'
intf eth0
```

```
# loop 5 times
loop 5
```

```
# send traffic 2x as fast
multiplier 2
```

```
# pad any packets out to their original size if they were truncated during capture
untruncate pad
```

You would then execute:

```
tcpreplay -f myconfigfile sample.pcap
```

⁶Looping files resets internal counters which control the speed that the file is replayed. Also because the file has to be closed and re-opened, an added delay between the last and first packet may occur.

Part III

Advanced Usage

4 Output: Interfaces, Packets & Files

4.1 Replaying on multiple interfaces

Tcpreplay can also split traffic so that each side of a connection is sent out a different interface⁷. In order to do this, tcpreplay needs the name of the second interface (-j) and a way to split the traffic. Currently, there are two ways to split traffic:

1. -C = split traffic by source IP address which is specified in CIDR notation
2. -c = split traffic according to a tcpprep cachefile⁸

When splitting traffic, it is important to remember that traffic that matches the filter is sent out the primary interface (-i). In this case, when splitting traffic by source IP address, you provide a list of networks in CIDR notation. For example:

- To send traffic from 10.0.0.0/8 out eth0 and everything else out eth1:
tcpreplay -C 10.0.0.0/8 -i eth0 -j eth1 sample.pcap
- To send traffic from 10.1.0.0/24 and 10.2.0.0/20 out eth0 and everything else out eth1:
tcpreplay -C 10.1.0.0/24,10.2.0.0/20 -i eth0 -j eth1 sample.pcap
- After using tcpprep to generate a cache file, you can use it to split traffic between two interfaces like this:
tcpreplay -c sample.cache -i eth0 -j eth1 sample.pcap

4.2 Selectively sending or dropping packets

Sometimes, you want to do some post-capture filtering of packets. Tcpreplay let's you have some control over which packets get sent.

1. -M = disables sending of martian packets. By definition, martian packets have a source IP of 0.x.x.x, 127.x.x.x, or 255.x.x.x
2. -x = send packets which match a specific pattern
3. -X = send packets which do not match a specific pattern

Both -x and -X support a variety of pattern matching types. These types are specified by a single character, followed by a colon, followed by the pattern. The following pattern matching types are available:

1. S - Source IP
Pattern is a comma delimited CIDR notation
2. D - Destination IP
Pattern is a comma delimited CIDR notation
3. B - Both source and destination IP must match
Pattern is a comma delimited CIDR notation
4. E - Either source or destination IP must match
Pattern is a comma delimited CIDR notation
5. P - A list of packet numbers from the pcap file.
Pattern is a series of numbers, separated by commas or dashes.
6. F - BPF syntax (same as used in tcpdump).
Filter must be quoted and is only supported with -x⁹.

⁷Note that you can also use the following options to split traffic into two files using -w and -W which are described later on in this FAQ.

⁸For information on generating tcpprep cache files, see the section on tcpprep.

⁹Note that if you want to send all the packets which do not match a bpf filter, all you have to do is negate the bpf filter. See the tcpdump(1) man page for more info.

Examples:

- To only send traffic that is too and from a host in 10.0.0.0/8:
tcpreplay -x B:10.0.0.0/8 -i eth0 sample.pcap
- To not send traffic that is too or from a host in 10.0.0.0/8:
tcpreplay -X E:10.0.0.0/8 -i eth0 sample.pcap
- To send every packet except the first 10 packets:
tcpreplay -X P:1-10 -i eth0 sample.pcap
- To only send the first 50 packets followed by packets: 100, 150, 200 and 250:
tcpreplay -x P:1-50,100,150,200,250 -i eth0 sample.pcap
- To only send TCP packets from 10.0.0.1:
tcpreplay -x F:'tcp and host 10.0.0.1' -i eth0 sample.pcap

4.3 Replaying only a few packets

Using the limit packets flag (-L) you can specify that tcpreplay will only send at most a specified number of packets.

- To send at most 100 packets:
tcpreplay -i eth0 -L 100 sample.pcap

4.4 Skipping the first bytes in a pcap file

If you want to skip the beginning of a pcap file, you can use the offset flag (-o) to skip a specified number of bytes and start sending on the next packet.

- To skip 15Kb into the pcap file and start sending packets from there:
tcpreplay -i eth0 -o 15000 sample.pcap

4.5 Replaying packets which are bigger then the MTU

Occasionally, you might find yourself trying to replay a pcap file which contains packets which are larger then the MTU for the sending interface. This might be due to the packets being captured on the loopback interface or on a 1000Mbps ethernet interface supporting “jumbo frames”. I’ve even seen packets which are 1500 bytes but contain both an ethernet header and trailer which bumps the total frame size to 1518 which is 4 bytes too large.

By default, tcpreplay will skip these packets and not send them. Alternatively, you can specify the -T flag to truncate these packets to the MTU and then send them. Of course this may invalidate your testing, but it has proven useful in certain situations. Also, when this feature is enabled, tcpreplay will automatically recalculate the IP and TCP, UDP or ICMP checksums as needed. Example:

```
tcpreplay -i eth0 -T sample.pcap
```

4.6 Writing packets to a file

It’s not always necessary to write packets to the network. Since tcpreplay has so many features which modify and select which packets are sent, it is occasionally useful to save these changes to another pcap file for comparison. Rather than running a separate tcpdump process to capture the packets, tcpreplay now supports output directly to a file. Example:

```
tcpreplay -i eth0 -w output.pcap -F -u pad -x E:10.0.0.0/8 input1.pcap input2.pcap input3.pcap
```

Notice that specifying an interface is still required (required for various internal functions), but all the packets will be written to *output.pcap*.

You can also split traffic into two files by using -W <2nd output file>.

4.7 Extracting Application Data (Layer 7)

New to version 2.0 is the ability to extract the application layer data from the packets and write them to a file. In the man page, we call this “data dump mode” which is enabled with `-D`. It’s important to specify `-D` before `-w` (and `-W` if you’re splitting data into two files). Example:

```
tcpreplay -D -i eth0 -j eth0 -w clientdata -W serverdata -C 10.0.0.0/24 sample.pcap
```

4.8 Replaying Live Traffic

You can now replay live traffic sniffed on one network interface and replay it on another interface using the `-S` flag to indicate sniff mode and the appropriate snaplen in bytes (0 denotes the entire packet). You can also enabling bi-directional traffic using the bridge mode flag: `-b`.

NOTE: It is critical for your sanity (and to prevent your murder by your network administrators) that the input interface and the output interface be on separate networks and additionally that no other network devices (such as bridges, switches, routers, etc) be connecting the two networks, else you will surely get a networkstorm the likes that have not been seen for years.

- Send packets sniffed on eth0 out eth1:
tcpreplay -i eth1 -S 0 eth0
- Bridge two subnets connected to eth0 and eth1:
tcpreplay -i eth0 -j eth1 -b -S 0

By default, `tcpreplay` listens in promiscuous mode on the specified interface, however if you only want to send unicasts directed for the local system and broadcasts, you can specify the “not_nosy” option in the configuration file or `-n` on the command line. Note that if another program has already placed the interface in promiscuous mode, the `-n` flag will have no effect, so you may want to use the `-x` or `-X` argument to limit packets.

4.9 Replaying Packet Capture Formats Other Than Libpcap

There are about as many different capture file formats as there are sniffers. In the interest of simplicity, `tcpreplay` only supports libpcap¹⁰. If you would like to replay a file in one of these multitude of formats, the excellent open source tool Ethereal easily allows you to convert it to libpcap. For instance, to convert a file in Sun’s snoop format to libpcap, issue the command:

```
tethereal -r blah.snoop -w blah.pcap
```

and replay the resulting file.

4.10 Replaying Client Traffic to a Server

A common question on the `tcpreplay-users` list is how does one replay the client side of a connection back to a server. Unfortunately, `tcpreplay` doesn’t support this right now. The major problem concerns syncing up TCP Seq/Ack numbers which will be different. ICMP also often contains IP header information which would need to be adjusted. About the only thing that could be easy to do is UDP, which isn’t usually requested.

This is however a feature that we’re looking into implementing in the `flowreplay` utility. If you’re interested in helping work on this feature, please contact us and we’d be more than happy to work with you. At this time however, we don’t have an ETA when this will be implemented, so don’t bother asking.

4.11 Decoding Packets

If the `tcpdump` binary is installed on your system when `tcpreplay` is compiled, it will allow you to decode packets as they are sent without running `tcpdump` in a separate window or worrying about it capturing packets which weren’t sent by `tcpreplay`.

- Decode packets as they are sent:
tcpreplay -i eth0 -v sample.pcap
- Decode packets with the link level header:
tcpreplay -i eth0 -v -A “-e” sample.pcap

¹⁰Note that some versions of `tcpreplay` prior to 1.4 also supported the Solaris snoop format.

- Fully decode and send one packet at a time:
tcpreplay -i eth0 -v -I -A "-s0 -evvvxX" sample.pcap

Note that tcpreplay automatically applies the `-n` flag to disable DNS lookups which would slow down tcpdump too much to make it effective.

5 Packet Editing

5.1 Rewriting MAC addresses

If you ever want to send traffic to another device on a switched LAN, you may need to change the destination MAC address of the packets. Tcpreplay allows you to set the destination MAC for each interface independently using the `-I` and `-J` switches. As of version 2.1.0, you can also specify the source MAC via `-k` and `-K`. Example:

- To send traffic out eth0 with a destination MAC of your router (00:00:01:02:03:04) and the source MAC of the server (00:20:30:40:50:60):
tcpreplay -i eth0 -I 00:00:01:02:03:04 -k 00:20:30:40:50:60 sample.pcap
- To split traffic between internal (10.0.0.0/24) and external addresses and to send that traffic to the two interfaces of a firewall:
tcpreplay -i eth0 -j eth1 -I 00:01:00:00:AA:01 -J 00:01:00:00:AA:02 -C 10.0.0.0/24 sample.pcap

5.2 Randomizing IP addresses

Occasionally, it is necessary to have tcpreplay rewrite the source and destination IP addresses, yet maintain the client/server relationship. Such a case might be having multiple copies of tcpreplay running at the same time using the same pcap file while trying to stress test firewall, IDS or other stateful device. If you didn't change the source and destination IP addresses, the device under test would get confused since it would see multiple copies of the same connection occurring at the same time. In order to accomplish this, tcpreplay accepts a user specified seed which is used to generate pseudo-random IP addresses. Also, when this feature is enabled, tcpreplay will automatically recalculate the IP and TCP, UDP or ICMP checksums as needed. Example:

```
tcpreplay -i eth0 -s 1239 sample.pcap &
tcpreplay -i eth0 -s 76 sample.pcap &
tcpreplay -i eth0 -s 239 sample.pcap &
tcpreplay -i eth0 sample.pcap
```

5.3 Replaying (de)truncated packets

Occasionally, it is necessary to replay traffic which has been truncated by tcpdump. This occurs when the tcpdump snaplen is smaller than the actual packet size. Since this will create problems for devices which are expecting a full-sized packet or attempting checksum calculations, tcpreplay allows you to either pad the packet with zeros or reset the packet length in the headers to the actual packet size. In either case, the IP and TCP, UDP or ICMP checksums are recalculated. Examples:

- Pad truncated packets:
tcpreplay -i eth0 -u pad sample.pcap
- Rewrite packet header lengths to the actual packet size:
tcpreplay -i eth0 -u trunc sample.pcap

5.4 Rewriting Layer 2 with -2

Starting in the 2.0.x branch, tcpreplay can replace the existing layer 2 header with one of your choosing. This is useful for when you want to change the layer 2 header type or add a header for pcap files without one. Each pcap file tells the type of frame. Currently tcpreplay knows how to deal with the following pcap(3) frame types:

- `DLT_EN10MB`
Replace existing 802.3/Ethernet II header
- `DLT_RAW`
Frame has no Layer 2 header, so we can add one.

- **DLT_LINUX_SLL**

Frame uses the Linux Cooked Socket header which is most commonly created with *tcpdump -i any* on a Linux system.

Tcpreplay accepts the new Layer 2 header as a string of comma separated hex values such as: 0xff,0xac,0x00,0x01,0xc0,0x64. Note that the leading '0x' is *not* required.

Potential uses for this are to add a layer 2 header for DLT_RAW captures or add/remove ethernet tags or QoS features.

5.5 Rewriting DLT_LINUX_SLL (Linux Cooked Socket) captures

Tcpdump uses a special frame type to store captures created with the “-i any” argument. This frame type uses a custom 16 byte layer 2 header which tracks which interface captured the packet and often the source MAC address of the original ethernet frame. Unfortunately, it never stores the destination MAC address and it doesn't store a source MAC when the packet is captured on the loopback interface. Normally, tcpreplay can't replay these pcap files because there isn't enough information in the LINUX_SLL header to do so; however two options do exist:

1. You can send these packets with -2 which will replace the LINUX_SLL header with an ethernet header of your choosing.
2. You can specify a destination MAC via -I and -J in which case tcpreplay will use the stored source MAC and create a new 802.3 Ethernet header. Note that if the pcap contains loopback packets, you will also need to specify -k and/or -K to specify the source MAC as well or they will be skipped.

5.6 Rewriting IP Addresses (pseudo-NAT)

Pseudo-NAT allows the mapping of IP addresses in IPv4 and ARP packets from one subnet to another subnet of the same or different size. This allows some or all the traffic sent to appear to come from a different IP subnet than it actually was captured on.

The mapping is done through a user specified translation table comprised of one or more source and destination network(s) in the format of <srcnet>/<masklen>:<dstnet>/<masklen> delimited by a comma. Mapping is done by matching IP addresses to the source subnet and rewriting the most significant bits with the destination subnet. For example:

```
tcpreplay -i eth0 -N 10.100.0.0/16:172.16.10.0/24 sample.pcap
```

would match any IP in the 10.100.0.0/16 subnet and rewrite it as if it came from or sent to the 172.16.10.0/24 subnet. Ie: 10.100.5.88 would become 172.16.10.88 and 10.100.99.45 would become 172.16.10.45. But 10.150.7.44 would not be rewritten.

For any given IP address, the translation table is applied in order (so if there are multiple mappings, earlier maps take precedence) and occurs only once per IP (no risk of an address getting rewritten a second time).

5.7 Advanced pseudo-NAT

Pseudo-NAT also works with traffic splitting (using two interfaces or output files) but with a few important differences. First you have the option of specifying one or two pseudo-NAT tables. Using a single pseudo-NAT table means that the source and destination IP addresses of both interfaces are rewritten using the same rules. Using two pseudo-NAT tables (specifying -N <Table1> -N <Table2>) will cause the source and destination IP addresses to be rewritten differently for each interface using the following matrix:

	Out Primary Interface	Out Secondary Interface
Src IP	Table 1	Table 2
Dest IP	Table 2	Table 1

While seemingly a bit confusing, this feature provides a number of interesting possibilities such as the ability to rewrite the IP headers of packets in the case where traffic is captured on the loopback interface (and the source and destination address is always 127.0.0.1) so that tcpreplay can make it look like two different systems are talking to each other (you'll probably also need to specify the source and destination MAC addresses via -I, -J, -k and -K).

5.8 IP Endpoints

While pseudo-NAT provides a great deal of flexibility, it is often more complicated than is necessary for testing of inline devices. As a simpler alternative, tcpreplay supports the concept of rewriting all traffic to so that it appears to be between two IP addresses:

```
tcpreplay -i eth0 -j eth1 -c sample.cache -e 10.0.0.1:10.1.1.1 sample.pcap
```

Will rewrite all the traffic so that it is between 10.0.0.1 and 10.1.1.1. The equivalent command using -N would be:

```
tcpreplay -i eth0 -j eth1 -c sample.cache -N 0.0.0.0/0:10.0.0.1 -N 0.0.0.0/0:10.1.1.1 sample.pcap
```

5.9 Unifying Dual-Outputs

Since a number of tcpreplay's packet editing functions require splitting traffic between client and servers, one problem that may arise is needing to edit packets but still output to a single interface or file. The solution to this is to use the one output option `-O` which causes packets to be processed as if they will be split between the interfaces/files, but then always go out the primary interface or file. Note that even though only one interface/file will be written to, both `-i` and `-j` must be specified; although they can be the same physical interface.

```
tcpreplay -i eth0 -j eth0 -O -c sample.cache -e 10.0.0.1:10.1.1.1 sample.pcap
```

Merging the output to a single file:

```
tcpreplay -i eth0 -j eth0 -w rewrite.pcap -c sample.cache -e 10.0.0.1:10.1.1.1 sample.pcap
```

6 Tcpprep Usage

6.1 What is tcpprep?

Tcpreplay can send traffic out two network cards, however it requires the calculations be done in real-time. These calculations can be expensive and can significantly reduce the throughput of tcpreplay.

Tcpprep is a libpcap pre-processor for tcpreplay which enables using two network cards to send traffic without the performance hit of doing the calculations in real-time.

6.2 What are these 'modes' tcpprep has?

Tcpprep has three basic modes which require the user to specify how to split traffic.

- CIDR (`-c`) mode requires the user to provide a list of networks. Any packet with a source IP in one of these networks gets sent out the primary interface.
- Regex (`-r`) mode requires the user to provide a regular expression. Any packet with a source IP matching the regex gets sent out the primary interface.
- Port (`-p`) mode splits TCP/UDP traffic based on the destination port in the header. Normally, ports 0-1023 are considered "server" ports and everything else a client port. You can create your own custom mapping file in the same format as `/etc/services` (see the `services(5)` man page for details) by specifying `-s <file>`.

And four auto modes in which tcpprep decides how to split traffic. Auto modes are useful for when you don't know much about the contents of the dump file in question and you want to split traffic up based upon servers and clients.

- Auto/Router (`-a -n router`) mode tries to find the largest network(s) that contain all the servers and no clients. Any unknown system is automatically re-classified as servers if it's inside the server network(s), otherwise it is classified as a client.
- Auto/Bridge (`-a -n bridge`) mode makes the assumption that the clients and servers are horribly intermixed on the network and there's no way to subnet them. While this takes less processing time to create the cache file it is unable to deal with unknown systems.
- Auto/Client (`-a -n client`) mode which works just like Auto/Bridge mode, except that any system it can't figure out is treated like a client.
- Auto/Server (`-a -n server`) mode which works just like Auto/Bridge mode, except that any system it can't figure out is treated like a server.

6.3 Splitting traffic based upon IP address

Tcpprep supports the same CIDR mode that tcpreplay supports using the `-c` flag (tcpreplay uses `-C`). Additionally, tcpprep also supports `regex(7)` regular expressions to match source IP addresses using the `-r` flag.

6.4 Auto Mode

6.4.1 How does Auto/Bridge mode work?

Tcpprep does an initial pass over the libpcap file to build a binary tree (one node per IP). For each IP, it keeps track of how many times it was a client or server. It then does a second pass of the file using the data in the tree and the ratio to determine if an IP is a client or server. If tcpprep is unable to determine the type (client or server) for each and every packet, then auto/bridge mode will fail. In these cases, it is best to use a different auto mode.

6.4.2 How does Auto/Router mode work?

Tcpprep does the same first pass as Auto/Bridge mode. It then tries to convert the binary tree into a list of networks containing the servers. Finally it uses the CIDR mode with the list of server networks in a second pass of the libpcap file. Unlike auto/bridge mode, auto/router mode can always successfully split IP addresses into clients and servers.

6.4.3 Determining Clients and Servers

Tcpprep uses the following methods in auto/router and auto/bridge mode to determine if an IP address is a client or server:

- Client:
 - TCP with Syn flag set
 - UDP source/destination port 53 (DNS) without query flag set
 - ICMP port unreachable (destination IP of packet)
- Server:
 - TCP with Syn/Ack flag set
 - UDP source/destination port 53 (DNS) with query flag set
 - ICMP port unreachable (source IP of packet)

6.4.4 Client/Server ratio

Since a system may send traffic which would classify it as both a client and server, it's necessary to be able to weigh the traffic. This is done by specifying the client/server ratio (-R) which is by default set to 2.0. The ratio is the modifier to the number of client connections. Hence, by default, client connections are valued twice as high as server connections.

6.5 Selectively sending/dropping packets

Tcpprep supports the same -x and -X options to selectively send or drop packets.

6.6 Using tcpprep cache files with tcpreplay

Just run:

```
tcpreplay -c sample.cache -i eth0 -j eth1 sample.pcap
```

6.7 Commenting tcpprep cache files

In versions of tcpprep \geq 2.1.0, you can specify a comment to be embedded in the tcpprep cache file. Comments are user specified and automatically include the command line arguments passed to tcpprep.

```
tcpprep -C "this is my comment" -i sample.pcap -o sample.cache <other args>
```

Or for no user comment, but still embed the command arguments:

```
tcpprep -C "" -i sample.pcap -o sample.cache <other args>
```

You can then later on print out the comments by running:

```
tcpprep -P sample.cache
```

7 Flowreplay Usage

While tcreplay is a great way to test NIDS and firewalls, it can't be used to test servers or HIDS since tcreplay can't connect to a service running on a device. The solution to this problem is flowreplay which instead of sending packets at Layer 2 (ethernet header and up), it can actually connect via TCP or UDP to server and then sends and receives data based upon a pcap capture file created with a tool like Ethereal or tcpdump.

Please note that flowreplay is currently alpha quality and is missing a number of key features.

7.1 How flowreplay works

Put simply, flowreplay opens a socket connection to a service on a target system(s) and sends data over that socket based on the packet capture. Flowreplay has no understanding of the application protocol (like HTTP or FTP) so it is somewhat limited in how it can deal with complicated exchanges between client and server.

Some of these limitations are:

- Flowreplay only plays the client side¹¹ of the connection.
- Flowreplay doesn't understand the application protocols. Hence it can't always deal with the case when the server sends a different response than what was originally captured in the pcap file.
- Flowreplay only sends TCP and UDP traffic.
- Flowreplay doesn't know about multi-flow protocols like FTP.
- Flowreplay can't listen on a port and wait for a client to connect to it.

7.2 Running flowreplay

See the flowreplay(8) man page for details.

8 Tuning OS's for high performance

Regardless of the size of physical memory, UNIX kernels will only allocate a static amount for network buffers. This includes packets sent via the "raw" interface, like with tcreplay. Most kernels will allow you to tweak the size of these buffers, drastically increasing performance and accuracy.

NOTE: The following information is provided based upon our own experiences or the reported experiences of others. Depending on your hardware and specific hardware, it may or may not work for you. It may even make your system horribly unstable, corrupt your harddrive, or worse.

NOTE: Different operating systems, network card drivers, and even hardware can have an effect on the accuracy of packet timestamps that tcpdump or other capture utilities generate. And as you know: garbage in, garbage out.

NOTE: If you have information on tuning the kernel of an operating system not listed here, please send it to me so I can include it.

8.1 Linux 2.4.x

The following is known to apply to the 2.4.x series of kernels. If anyone has any information regarding other kernel versions, please let us know. By default Linux's tcreplay performance isn't all that stellar. However, with a simple tweak, relatively decent performance can be had on the right hardware. By default, Linux specifies a 64K buffer for sending packets. Increasing this buffer to about half a megabyte does a good job:

```
echo 524287 >/proc/sys/net/core/wmem_default
echo 524287 >/proc/sys/net/core/wmem_max
echo 524287 >/proc/sys/net/core/rmem_max
echo 524287 >/proc/sys/net/core/rmem_default
```

On one system, we've seen a jump from 23.02 megabits/sec (5560 packets/sec) to 220.30 megabits/sec (53212 packets/sec) which is nearly a 10x increase in performance. Depending on your system and capture file, different numbers may provide different results.

¹¹Flowreplay assumes the first UDP packet on a given 4-tuple is the client

8.2 *BSD

*BSD systems typically allow you to specify the size of network buffers with the NMBCLUSTERS option in the kernel config file. Experiment with different sizes to see which yields the best performance. See the options(4) man page for more details.

9 Understanding Common Error and Warning Messages

9.1 Can't open eth0: libnet_select_device(): Can't find interface eth0

Generally this occurs when the interface (eth0 in this example) is not up or doesn't have an IP address assigned to it.

9.2 Can't open lo: libnet_select_device(): Can't find interface lo

Version 1.1.0 of Libnet is unable to send traffic on the loopback device. Upgrade to a later release of the Libnet library to solve this problem.

9.3 Can't open eth0: UID != 0

Tcpdump requires that you run it as root.

9.4 100000 write attempts failed from full buffers and were repeated

When tcpdump displays a message like "100000 write attempts failed from full buffers and were repeated", this usually means the kernel buffers were full and it had to wait until memory was available. This is quite common when replaying files as fast as possible with the "-R" option. See the tuning OS section in this document for suggestions on solving this problem.

9.5 Invalid mac address: 00:00:00:00:00:00

Currently tcpdump reserves the MAC address of 00:00:00:00:00:00 as reserved for internal use. Hence you can't rewrite the MAC address of packets to be all zeros. While we intend to fix this someday it's not currently high on our priority list, so let us know if we should re-prioritize things.

9.6 Unable to process test.cache: cache file version mismatch

Cache files generated by tcpdump and read by tcpdump are versioned to allow enhancements to the cache file format. Anytime the cache file format changes, the version is incremented. Since this occurs on a very rare basis, this is generally not an issue; however anytime there is a change, it breaks compatibility with previously created cache files. The solution for this problem is to use the same version of tcpdump and tcpdump to read/write the cache files. Cache file versions match the following versions of tcpdump/tcpdump:

- Version 1:
Prior to 1.3.beta1
- Version 2:
1.3.beta2 to 1.3.1/1.4.beta1
- Version 3:
1.3.2/1.4.beta2 to 2.0.3
- Version 4:
2.1.0 and above. Note that prior to version 2.3.0, tcpdump had a bug which broke cache file compatibility between big and little endian systems.

9.7 Skipping SLL loopback packet.

Your capture file was created on Linux with the 'any' parameter which then captured a packet on the loopback interface. However, tcpdump doesn't have enough information to actually send the packet, so it skips it. Specifying a source and destination MAC address (-I, -k, -J, -K) will allow tcpdump to send these packets.

9.8 Packet length (8892) is greater than MTU; skipping packet.

The packet length (in this case 8892 bytes) is greater than the maximum transmission unit (MTU) on the outgoing interface. Tcpreplay must skip the packet. Alternatively, you can specify the -T option and tcpreplay will truncate the packet to the MTU size, fix the checksums and send it.

9.9 Why is tcpreplay not sending all the packets?

Every now and then, someone emails the tcpreplay-users list, asking if there is a bug in tcpreplay which causes it not to send all the packets. This usually happens when the user uses the -R flag or is replaying a high-speed pcap file (> 50Mbps, although this number is dependant on the hardware in use).

The short version of the answer is: no, we are not aware of any bugs which might cause a few packets to not be sent.

The longer version goes something like this:

If you are running tcpreplay multiple times and are using tcpdump or other packet sniffer to count the number packets sent and are getting different numbers, it's not tcpreplay's fault. The problem lies in one of two places:

1. It is well known that tcpdump and other sniffers have a problem keeping up with high-speed traffic. Furthermore, the OS in many cases *lies* about how many packets were dropped. Tcpdump will repeat this lie to you. In other words, tcpdump isn't seeing all the packets. Usually this is a problem with the network card or driver which may or may not be fixable. Try another network card/driver.
2. When tcpreplay sends a packet, it actually gets copied to a send buffer in the kernel. If this buffer is full, the kernel is supposed to tell tcpreplay that it didn't copy the packet to this buffer. If the kernel has a bug which squelches this error, tcpreplay will not keep trying to send the packet and will move on to the next one. Currently I am not aware of any OS kernels with this bug, but it is possible that it exists. If you find out that your OS has this problem, please let me know so I can list it here.

If for some reason, you still think its a bug in tcpreplay, by all means read the code and tell me how stupid I am. The do_packets() function in do_packets.c is where tcpreplay processes the pcap file and sends all of the packets.

10 Required Libraries and Tools

10.1 Libpcap

As of tcpreplay v1.4, you'll need to have libpcap installed on your system. As of v2.0, you'll need at least version 0.6.0 or better, but I only test our code with the latest version. Libpcap can be obtained on the tcpdump homepage¹².

10.2 Libnet

Tcpreplay v1.3 is the last version to support the old libnet API (everything before 1.1.x). As of v1.4 you will need to use Libnet 1.1.0 or better which can be obtained from the Libnet homepage¹³.

10.3 Libpcapnav

Starting with v2.0, tcpreplay can use libpcapnav to support the jump offset feature. If libpcapnav is not found on the system, that feature will be disabled. Libpcapnav can be found on the NetDude homepage¹⁴.

¹²<http://www.tcpdump.org/>

¹³<http://www.packetfactory.net/Projects/Libnet/>

¹⁴<http://netdude.sourceforge.net/>

10.4 Tcpcap

As of 2.0, tcpreplay uses tcpdump (the binary, not code) to decode packets to STDOUT in a human readable (with practice) format as it sends them. If you would like this feature, tcpdump must be installed on your system.

NOTE: The location of the tcpdump binary is hardcoded in tcpreplay at compile time. If tcpdump gets renamed or moved, the feature will become disabled.

Part IV

Other Resources

11 Other pcap tools available

11.1 Tools to capture network traffic or decode pcap files

- tcpdump
<http://www.tcpdump.org/>
- ethereal
<http://www.ethereal.com/>
- ettercap
<http://ettercap.sourceforge.net/>

11.2 Tools to edit pcap files

- tcpslice
Splits pcap files into smaller files
<http://www.tcpdump.org/>
- mergcap
Merges two pcap capture files into one
<http://www.ethereal.com/>
- pcapmerge
Merges two or more pcap capture files into one
<http://tcpreplay.sourceforge.net/>
- editcap
Converts capture file formats (pcap, snoop, etc)
<http://www.ethereal.com/>
- netdude
GTK based pcap capture file editor. Allows editing most anything in the packet.
<http://netdude.sourceforge.net/>

11.3 Other useful tools

- capinfo
Prints statistics and basic information about a pcap file
<http://tcpreplay.sourceforge.net/>
- text2pcap
Generates a pcap capture file from a hex dump
<http://www.ethereal.com/>
- tcpflow
Extracts and reassembles the data portion on a per-flow basis on live traffic or pcap capture files
<http://www.circlemud.org/~jelson/software/tcpflow/>