

Tcpreplay 1.5.x FAQ

Aaron Turner <aturner_AT_pobox.com>
<http://tcpreplay.sourceforge.net/>

Last Edited:
Dec 10, 2003

1	General Info	4
1.1	What is this FAQ for?	4
1.2	What tools come with tcpreplay?	4
1.3	How can I get tcpreplay's source?	4
1.4	What requirements does tcpreplay have?	4
1.5	How do I compile tcpreplay?	4
1.6	Are there binaries available?	4
1.7	Is there a Microsoft Windows port?	5
1.8	How is tcpreplay licensed?	5
1.9	What are some uses for tcpreplay?	5
1.10	What are some uses for flowreplay?	5
2	Bugs, Feature Requests, and Patches	5
2.1	Where can I get help or report bugs?	5
2.2	What information should I provide when I report a bug?	5
2.3	I have a feature request, what should I do?	5
2.4	I've written a patch for tcpreplay, how can I submit it?	5
2.5	Patch requirements	6
3	Basic Tcpreplay Usage	6
3.1	What is tcpreplay?	6
3.2	What isn't tcpreplay?	6
3.3	Replaying the traffic	6
3.4	Replaying at different speeds	6
3.5	Replaying the same file over and over again	7
3.6	Replaying only a few packets	7
4	Advanced Tcpreplay Usage	7
4.1	Replaying on multiple interfaces	7
4.2	Selectively sending or dropping packets	7
4.3	Skipping the first bytes in a pcap file	8
4.4	Rewriting destination MAC address	8
4.5	Randomizing IP addresses	8
4.6	Replaying (de)truncated packets	9
4.7	Replaying packets which are bigger then the MTU	9
4.8	Writing packets to a file	9
4.9	Extracting Application Data (Layer 7)	9
4.10	Using Configuration Files	9
4.11	Replaying Live Traffic	10
4.12	Bridging Live Traffic	10
4.13	Replaying Packet Capture Formats Other Than Libpcap	10
4.14	Replaying Client Traffic to a Server	10
4.15	Rewriting Layer 2 with -2	11

5.1	What is tcpdump?	11
5.2	How does tcpdump work?	11
5.3	Does tcpdump modify my libpcap file?	11
5.4	Why use tcpdump?	11
5.5	Can a cache file be used for multiple (different) libpcap files?	11
5.6	Why would I want to use tcpdump with two network cards?	12
5.7	How big are the cache files?	12
5.8	What are these 'modes' tcpdump has?	12
5.9	Splitting traffic based upon IP address	12
5.10	Auto Mode	12
5.10.1	How does Auto/Bridge mode work?	12
5.10.2	How does Auto/Router mode work?	12
5.10.3	Determining Clients and Servers	13
5.10.4	Client/Server ratio	13
5.11	Selectively sending/dropping packets	13
5.12	Using tcpdump cache files with tcpdump	13
6	Flowreplay Usage	13
6.1	How flowreplay works	13
6.2	Running flowreplay	14
7	Tuning OS's for high performance	14
7.1	Linux 2.4.x	14
7.2	*BSD	14
8	Required Libraries	14
8.1	Libpcap	14
8.2	Libnet	14
8.3	Libpcapnav	14
9	Understanding Common Error Messages	15
9.1	Can't open eth0: libnet_select_device(): Can't find interface eth0	15
9.2	Can't open lo: libnet_select_device(): Can't find interface lo	15
9.3	Can't open eth0: UID != 0	15
9.4	100000 write attempts failed from full buffers and were repeated	15
9.5	Invalid mac address: 00:00:00:00:00:00	15
9.6	Unable to process test.cache: cache file version mismatch	15
9.7	Skipping SLL loopback packet.	15
9.8	Packet length (8892) is greater than MTU; skipping packet.	15
9.9	Why is tcpdump not sending all the packets?	16
10	Other pcap tools available	16
10.1	Tools to capture network traffic or decode pcap files	16
10.2	Tools to edit pcap files	16
10.3	Other tools	17
A	BSD License	18

1.1 What is this FAQ for?

Tcpreplay is a suite of powerful tools, but with that power comes complexity. While we have done our best to write good man pages for tcpreplay and it's associated utilities, we understand that many people may want more information than we can provide in the man pages. Additionally, this FAQ attempts to cover material which we feel will be of use to people using tcpreplay, as well as common questions that occur on the Tcpreplay-Users mailing list.

1.2 What tools come with tcpreplay?

- tcpreplay - replay ethernet packets stored in a pcap file as they were captured
- tcpdump - a pcap pre-processor for tcpreplay
- flowreplay - connects to a server(s) and replays the client side of the connection stored in a pcap file
- pcapmerge - merges two or more pcap files into one
- capinfo - displays basic information about a pcap file

1.3 How can I get tcpreplay's source?

The source code to tcpreplay is available on the tcpreplay homepage: <http://tcpreplay.sf.net/>

1.4 What requirements does tcpreplay have?

1. You'll need the libnet and libpcap libraries.
2. You'll also need a compatible operating system. Basically, any UNIX-like or UNIX-based operating system should work. Linux, *BSD, Solaris, OS/X and others should all work. If you find any compatibility issues with any UNIX-like/based OS, please let us know.

1.5 How do I compile tcpreplay?

Two easy steps:

1. As a normal user: *./configure && make*
2. As root: *make test -i && make install*

There are some optional arguments which can be passed to the configure script which may help in cases where your libnet or libpcap 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.

A few comments about 'make test':

- make test is just a series of sanity checks which try to find serious bugs (crashes) in tcpdump and tcpreplay.
- make test requires at least one properly configured network interface.
- If make test fails, 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.

1.6 Are there binaries available?

Occasionally. And even when we do, generally only for one or two operating systems. Generally speaking, we assume people who want to use a tool like this can figure out how to compile it.

Sorry, none of the developers have yet to sell their soul. Now if you want to port tcpreplay to Windows, let us know and we will try to help you.

1.8 How is tcpreplay licensed?

Tcpreplay is licensed under the BSD License. For details, see Appendix A.

1.9 What are some uses for tcpreplay?

Originally, tcpreplay was written to test network intrusion detection systems (NIDS), however tcpreplay has been used to test firewalls, routers, and other network devices.

1.10 What are some uses for flowreplay?

A lot of people wanted a tool like tcpreplay, but wanted to be able to replay traffic *to* a server. Since tcpreplay was unable to do this, I developed flowreplay which replays the data portion of the flow, but recreates the connection to the specified server(s). This makes flowreplay an ideal tool to test host intrusion detection systems (HIDS) as well as captured exploits and security patches when the actual exploit code is not available.

2 Bugs, Feature Requests, and Patches

2.1 Where can I get help or report bugs?

The best place to get help or report a bug is the Tcpreplay-Users mailing list:
<http://lists.sourceforge.net/lists/listinfo/tcpreplay-users>

2.2 What information should I provide when I report a bug?

One of the most frustrating things for any developer trying to help a user with a problem is not enough information. Please be sure to include *at minimum* the following information, however any additional information you feel may be helpful will be appreciated.

- Version information (output of -V)
- Command line used (options and arguments)
- Platform (Red Hat Linux 9 on Intel, Solaris 7 on SPARC, etc)
- Error message (if available) and/or description of problem
- If possible, attach the pcap file used (compressed with bzip2 or gzip preferred)

2.3 I have a feature request, what should I do?

Let us know! Many of the features exist today because users like you asked for them. To make a feature request, you can either email the tcpreplay-users mailing list (see above) or fill out the feature request form on the tcpreplay SourceForge website.

2.4 I've written a patch for tcpreplay, how can I submit it?

I'm always willing to include new features or bug fixes submitted by users. You may email me directly or the tcpreplay-users mailing list. Please *do not* use the Patch Tracker on the tcpreplay SourceForge web site.

- Be aware that submitting a patch, *you are licensing it under the BSD License* as written in Appendix A. If this is not acceptable to you, then *do not* send me the patch!
- If you wish to maintain the copyright over your code, be sure that your patch contains the appropriate information.
- Please provide a description of what your patch does!
- Comment your code! I won't use code I can't understand.
- Make sure you are patching a branch that is still being maintained. Generally that means that most recent stable and development branches (1.4 and 1.5 at the time of this writing).
- Make sure you are patching against the most recent release for that branch.
- Please submit your patch in the unified diff format so I can better understand what you're changing.
- Please provide any relevant personal information you'd like listed in the CREDITS file.

Please note that while I'm always interested in patches, I may rewrite some or all of your submission to maintain a consistent coding style.

3 Basic Tcpreplay Usage

3.1 What is tcpreplay?

In the simplest terms, tcpreplay is a tool to send network traffic stored in pcap format back onto the network; basically the exact opposite of tcpdump.

3.2 What isn't tcpreplay?

Tcpreplay is *not* a tool to replay captured traffic to a server or client. Specifically, tcpreplay does not have the ability to rewrite IP addresses to a user-specified value or synchronize TCP sequence and acknowledgment numbers. In other words, tcpreplay can't "connect" to a server or be used to emulate a server and have clients connect to it. If you're looking for that, check out flowreplay.

3.3 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
```

3.4 Replaying at different speeds

You can also replay the traffic at different speeds than it was originally captured¹. To support this, tcpreplay supports four different flags: -R, -r, -m, and -p

Some examples:

- 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`

¹Tcpreplay 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.

Using the loop flag (-l) you can specify that a pcap file will be sent two or more times²:

- 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`

3.6 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 Advanced Tcpreplay Usage

4.1 Replaying on multiple interfaces

Tcpreplay can also split traffic so that each side of a connection is sent out a different interface³. 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
2. -c = split traffic according to a tcpprep cachefile⁴

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 lets 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

²Looping 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.

³Note 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.

⁴For information on generating tcpprep cache files, see the section on tcpprep.

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, seperated by commas or dashes.
6. F - BPF syntax (same as used in tcpdump).
Filter must be quoted and is only supported with -x⁵.

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

4.4 Rewriting destination MAC address

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 independantly using the -I and -J swiches. Example:

- To send traffic out eth0 with a destination MAC of your router (00:00:01:02:03:04):
tcpreplay -i eth0 -I 00:00:01:02:03:04 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

4.5 Randomizing IP addresses

Occasionally, it is necessary to have tcprewrite rewrite the source and destination IP addresses, yet maintain the client/server relationship. Such a case might be having multiple copies of tcpdump 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 occuring 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 layer 3 and 4 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
```

⁵Note 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(8) man page for more info.

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 layer 3 and 4 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`

4.7 Replaying packets which are bigger than the MTU

Occasionally, you might find yourself trying to replay a pcap file which contains packets which are larger than 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. Example:

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

4.8 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>.

4.9 Extracting Application Data (Layer 7)

New to version 1.5 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
```

4.10 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
```

4.11 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.

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`

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.

4.12 Bridging Live Traffic

Much like replaying live traffic, `tcpreplay` can both listen and send on two interfaces, effectively connecting two broadcast domains by using the `-S` and `-b` flags.

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.

- Bridge two subnets connected to eth0 and eth1:
`tcpreplay -i eth0 -j eth1 -b -S 0`

4.13 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 `libpcap`⁶. 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.

4.14 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<<<<<< FAQ.lyx as well in many cases. About the only thing that could be easy to do is UDP, which isn’t usually requested.===== in many cases. About the only thing that could be easy to do is UDP which isn’t usually requested.>>>>>> 1.14

This is however a feature that we’re looking into implementing. If you’re interested in helping add 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.

⁶Note that some versions of `tcpreplay` prior to 1.4 also supported the Solaris snoop format.

Starting in the 1.5.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 used with *tcpdump -i any* on a Linux system. This uses a 16 byte Layer 2 header which contains the protocol type and source MAC address but *not* the destination MAC. You can send these packets with either *-2* or by specifying a destination MAC via *-I* and *-J* in which case tcpreplay will rewrite it as a standard 802.3 Ethernet header.

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 Tcpprep Usage

5.1 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.

5.2 How does tcpprep work?

Tcpprep reads in a libpcap (tcpdump) formatted capture file and does some processing to generate a tcpreplay cache file. This cache file tells tcpreplay which interface a given packet should be sent out of.

5.3 Does tcpprep modify my libpcap file?

No.

5.4 Why use tcpprep?

There are three major reasons to use tcpprep:

1. Tcpprep can split traffic based upon more methods and criteria than tcpreplay.
2. By pre-processing the pcap, tcpreplay has a higher theoretical maximum throughput.
3. By pre-processing the pcap, tcpreplay can be more accurate in timing when replaying traffic at normal speed.

5.5 Can a cache file be used for multiple (different) libpcap files?

Cache files have nothing linking them to a given libpcap file, so there is nothing to stop you from doing this. However running tcpreplay with a cache file from a different libpcap source file is likely to cause a lot of problems and is not supported.

Tcpreplay traditionally is good for putting traffic on a given network, often used to test a network intrusion detection system (NIDS). However, there are cases where putting traffic onto a subnet in this manner is not good enough- you have to be able to send traffic *through* a device such as a router, firewall, or bridge.

In these cases, being able to use a single source file (libpcap) for both ends of the connection solves this problem.

5.7 How big are the cache files?

Very small. Actual size depends on the number of packets in the dump file. Two bits of data is stored for each packet. On a test using a 900MB dump file containing over 500,000 packets, the cache file was only 150K.

5.8 What are these 'modes' tcpprep has?

Tcpprep has two 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.

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.

5.9 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.

5.10 Auto Mode

5.10.1 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.

5.10.2 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.

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)

5.10.4 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.

5.11 Selectively sending/dropping packets

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

5.12 Using tcpprep cache files with tcpreplay

Just run:

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

6 Flowreplay Usage

While tcpreplay is a great way to test NIDS and firewalls, it can't be used to test servers or HIDS since tcpreplay 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.

6.1 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 side⁷ 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.

⁷Flowreplay assumes the first UDP packet on a given 4-tuple is the client

7 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 tcpreplay. Most kernels will allow you to tweak the size of these buffers, drastically increasing performance and accuracy.

NOTE: Please note that 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.

7.1 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 tcpreplay 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.

7.2 *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.

8 Required Libraries

8.1 Libpcap

As of tcpreplay v1.4, you'll need to have libpcap installed on your system. As of v1.5, you'll need at least version 0.6.0 or better, but I only test our code with the latest version: 0.7.2. Libpcap can be obtained here: [Tcpdump http://www.tcpdump.org/](http://www.tcpdump.org/)

8.2 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 homepage: [Libnet http://www.packetfactory.net/Projects/Libnet/](http://www.packetfactory.net/Projects/Libnet/)

8.3 Libpcapnav

Starting with v1.5.alpha5, tcpreplay requires libpcapnav. I'm unsure if this requirement will continue when 1.5 goes beta or not. If this is a real problem for you, let me know and I'll take that into consideration. Libpcapnav can be found on the NetDude homepage: [NetDude http://netdude.sf.net/](http://netdude.sf.net/)

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. For example:
ifconfig eth0 1.1.1.1 netmask 255.255.255.0 up

Will do the trick on most operating systems.

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. Try upgrading to the 1.1.1 (pre)release of the Libnet library to solve this problem.

9.3 Can't open eth0: UID != 0

Tcpreplay requires that you run it as root.

9.4 100000 write attempts failed from full buffers and were repeated

When tcpreplay 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 tcpreplay reserves the MAC address of 00:00:00:00:00:00 as reserved for internal use. Hence you can't rewrite the destination 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 tcpprep and read by tcpreplay 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 tcpreplay and tcpprep to read/write the cache files. Cache file versions match the following versions of tcpprep/tcpreplay:

- 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 and above

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, tcpreplay doesn't have enough information to actual send the packet, so it skips it.

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

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

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 Other pcap tools available

10.1 Tools to capture network traffic or decode pcap files

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

10.2 Tools to edit pcap files

- tcpslice
Splits pcap files into smaller files
tcpdump
- mergecap
Merges two pcap capture files into one
Ethereal
- pcapmerge
Merges two or more pcap capture files into one
tcpdump
- editcap
Converts capture file formats (pcap, snoop, etc)
Ethereal
- netdude
GTK based pcap capture file editor. Allows editing most anything in the packet.
NetDude

- capinfo
Prints statistics and basic information about a pcap file
Tcpreplay
- text2pcap
Generates a pcap capture file from a hex dump
Ethereal

A BSD License

Copyright (c) 1999-2003 Matt Bing, Aaron Turner. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by Anzen Computing, Inc.
4. Neither the name of Anzen Computing, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.