Tcpreplay 1.4.x FAQ

Aaron Turner <aturner_AT_pobox.com>
Matt Bing <mbing_AT_nfr.net>
http://www.sourceforge.net/projects/tcpreplay/

FAQ Version 0.7

30th August 2003

Contents

Tcpreplay 1.4.x FAQ

1 General Info

1.1 What is this FAQ for?

Tcpreplay is a powerful tool, 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 then 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 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.

1.3 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.

1.4 How can I get tcpreplay's source?

The source code to tcpreplay is available on SourceForge: `http://www.sourceforge.net/projects/tcpreplay`

1.5 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 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.

1.6 How do I compile tcpreplay?

Two easy steps:

1. As a normal user: *./configure && make*
2. As root: *make test && 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 tcpprep 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.

1.7 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.

1.8 Is there a Microsoft Windows port?

Sorry, none of the developers have yet to sell their soul.

1.9 How is tcpreplay licensed?

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

1.10 What are some uses for tcpreplay?

Originally, tcpreplay was written to test network intrusion detection devices, however tcpreplay has been used to test firewalls, routers, and other in-line devices. In it's current form, tcpreplay is not very well suited for testing IP stacks or applications running on servers or clients since it is unable to synchronize TCP sequence numbers nor is able to react or use packets sent by other devices as input.

2 Undocumented Features (aka Bugs)

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 or description of problem
- If possible, attach the pcap file used (compressed with bzip or gzip preferred)

3 Basic 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 then it was originally captured[1]. 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 traffic at 10.5 packets/second:
  tcpreplay -p 10.5 -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 times[2]:

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

4 Advanced Usage

4.1 Replaying on multiple interfaces

Tcpreplay can also split traffic so that each side of a connection is sent out a different interface. 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 cachefile[3]

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

[1] Tcpreplay 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.

[2] Looping 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.

[3] For information on generating tcpprep cache files, see the section on tcpprep.

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.

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

4.3 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 independently using the -I and -J switches. 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

4.4 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 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

4.5 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 then 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

4.6 Using Configuration Files

4.7 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[4]. 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.

4.8 Replaying Client Traffic to a Server

A common question on the tcpreplay-users list is how does one 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 as well in many cases. About the only thing that could be easy to do is UDP which doesn't isn't usually requested.

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

5 Using tcpprep

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 tcpprep 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 then 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.

5.6 Why would I want to use tcpreplay with two network cards?

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. Four 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 300K.

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 two 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 trys 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.

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 auto/router mode.

5.10.2 How does Auto/Router mode work?

Tcpprep does the same first pass as Auto/Bridge mode. It then trys 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.

5.10.3 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)

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 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.

6.1 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, different numbers may provide different results.

6.2 *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.

7 Required Libraries

7.1 Libpcap

As of tcpreplay v1.4, you'll need to have Libpcap installed on your system. Any recent Libpcap version should do, but we only test our code with the latest version: 0.7.1. Libpcap can be obtained here: `http://www.tcpdump.org/`

7.2 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: `http://www.packetfactory.net/Projects/Libnet/`

8 Understanding Common Error Messages

8.1 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 UNIX and UNIX-like operating systems.

8.2 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.

8.3 Can't open eth0: UID != 0

Tcpreplay requires that you run it as root.

8.4 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.

8.5 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.

8.6 Unable to process test.cache: cache file version missmatch

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 Other pcap tools available

9.1 Tools to capture network traffic or decode pcap files

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

9.2 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

9.3 Other tools

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

Appendix

A BSD License