DNS-STATS Compactor User Guide

DNS-STATS

Version 1.2.3

Table of Contents

1. Overview

1.1. About

The DNS-STATS Compactor project is a suite of applications for capturing and working with DNS traffic to a DNS nameserver. It stores DNS traffic data in Compacted-DNS (C-DNS), a space-efficient format defined in RFC8618.

The project was initially developed for for ICANN by Sinodun IT, and is now released via DNS-STATS as an open source project licenced under the Mozilla Public License v2.0.

For more information about DNS-STATS and the Compactor see the DNS-STATS website.

1.2. DNS-STATS Compactor

The DNS-STATS Compactor suite currently comprises two programs:

- *compactor*. Similar in usage to the well-known *tcpdump* utility, *compactor* reads traffic from one or more network interfaces (or a DNSTAP socket) and writes selected details to C-DNS and PCAP output files. *compactor* can also read and convert pre-recorded PCAP files or DNSTAP files.
- *inspector*. Reconstructs network traffic from C-DNS files produced by *compactor*. It outputs one or more PCAP files suitable for direct inspection or input to existing analysis tools. See Reconstructed PCAP files for limitations on the reconstruction. Alternatively, *inspector* can be used to convert C-DNS files to text, based on a user-specified template.

compactor is resource efficient, and can therefore be co-located on a nameserver. Alternatively it can be run on a standalone server with access to the network traffic to be recorded.

compactor can be configured to produce multiple output files from a single data source. *compactor* can optionally compress output files using the popular *gzip* or *xz* compression schemes. The output file types that may be produced are:

- C-DNS. These contain captured DNS traffic, along with some ancillary information, e.g. ICMP and TCP Reset counts. These files are significantly smaller than PCAP files containing the same traffic. See C-DNS Format.
- 'Ignored' traffic. When capturing from the network, these contain captured non-DNS and malformed DNS packets in PCAP format. *Ignored* traffic is not available when capturing from DNSTAP.
- 'Raw' traffic. When capturing from the network, these contain all packets in the captured traffic in PCAP format. They are similar to files produced by *tcpdump*. *Raw* traffic is not available when capturing from DNSTAP.

1.3. C-DNS Format

Traditionally server operators and others wishing to record DNS traffic have used network level capture tools such as `tcpdump`. While this does produce a complete record of the traffic to and from the server, the resulting output files are large. As the files contain a lot of repeated data (e.g. server IP and MAC address, common port numbers), they compress well, typically reducing in size by an order of magnitude. This compression, however, requires notable CPU resources to perform.

The DNS-STATS Compactor focuses on the needs of DNS operators capturing data in environments where resources (CPU, Upload bandwidth, etc.) are restricted.

The C-DNS file format is designed for efficiently recording DNS traffic information:

- It only captures transport level information likely to be of interest to a DNS operator.
- It is highly flexible and can be configured to capture only specific pieces of data, basic query and response information, or additional details up to the entire DNS payload. See Configuring.
- It performs DNS-specific compression internally to produce files that are significantly smaller than raw PCAP files even when full payloads are captured.

1.3.1. C-DNS versioning

From version 1.0 DNS-STATS Compactor writes and reads captured traffic using the 1.0 C-DNS file format as defined in RFC8618.

Previous versions of the DNS-STATS Compactor implemented earlier, more contstrained versions of the draft specification. To support backwards compatibility *inspector* reads:

- Version 1.0 of C-DNS as described in RFC8618 and
- The two older formats (0.2 and 0.5) used by the previous *compactor* releases (see below for details of important differences).

It is intended to remain backwards compatible in any future format changes.

1.3.1.1. *compactor* private version and customisations

compactor 1.0 and later writes C-DNS as described in RFC8618, with the following changes:

- A private version ID of 3 is present.
- Some *compactor* implementation-specific entries are added to several C-DNS maps (see RFC section 7.1). They are listed below with their *compactor* key values in parenthesis.
  - *CollectionParameters*:
    - *compactor-dns-port* (-1): specifies the configured port on which *compactor* listened for traffic.
  - *BlockPreamble*:
    - *compactor-end-time* (-1): if the block rolled over (i.e. a new incoming data caused collection to begin a new block), the timestamp of that new data is recorded as the end time of the (older) block. If collection from an interface or DNSTAP stops, the time collection stops is recorded as the end time of the block. No end time is recorded when reading a PCAP or DNSTAP file.
    - *compactor-start-time* (-2): if the block rolled over (i.e. a new incoming data caused

collection to begin a new block), the timestamp of that new data is recorded as the start time of the (newer) block. If collection from an interface or DNSTAP starts, the time collection starts is recorded as the start time of the block. No start time is recorded when reading a PCAP or DNSTAP file.

- *BlockStatistics*:
  - *compactor-non-dns-packets* (-1): count of the number of received packets that could not be interpreted as DNS packets.
  - *compactor-out-of-order-packets* (-2): count of the number of received packets that were not received by *compactor* in strict chronological order.
  - *compactor-missing-pairs* (-3): count of output query/response pairs not written by *compactor* because they could not be processed quickly enough.
  - *compactor-missing-packets* (-4): count of output raw PCAP packets not written by *compactor* because they could not be processed quickly enough.
  - *compactor-missing-non-dns* (-5): count of output ignored PCAP packets not written by *compactor* because they could not be processed quickly enough.
  - *compactor-packets* (-6): total packets received by compactor
  - *compactor-missing-received* (-7): count of packets sniffed from the network not processed by *compactor* because they could not be processed quickly enough.
  - *compactor-discarded-packets* (-8): count of packets actively discarded by compactor due to some processing threshold, e.g. sampling.
  - *compactor-missing_matcher* (-9): count of packets unmatched packets not written by *compactor* because they could not be processed for matching quickly enough.
  - *pcap-packets* (-10): informational only report from pcap library - count of packets received
  - *pcap-missing-if* (-11): informational only report from pcap library - count of packets dropped at the interface
  - *pcap-missing-os* (-12): informational only report from pcap library - count of packets dropped in the kernel

The current release does not support the following facilities defined in the RFC:

- Malformed packet data recorded directly into C-DNS.
- `response-processing-data` field

In addition, note that the `qr-type` field is only present when reading from DNSTAP.

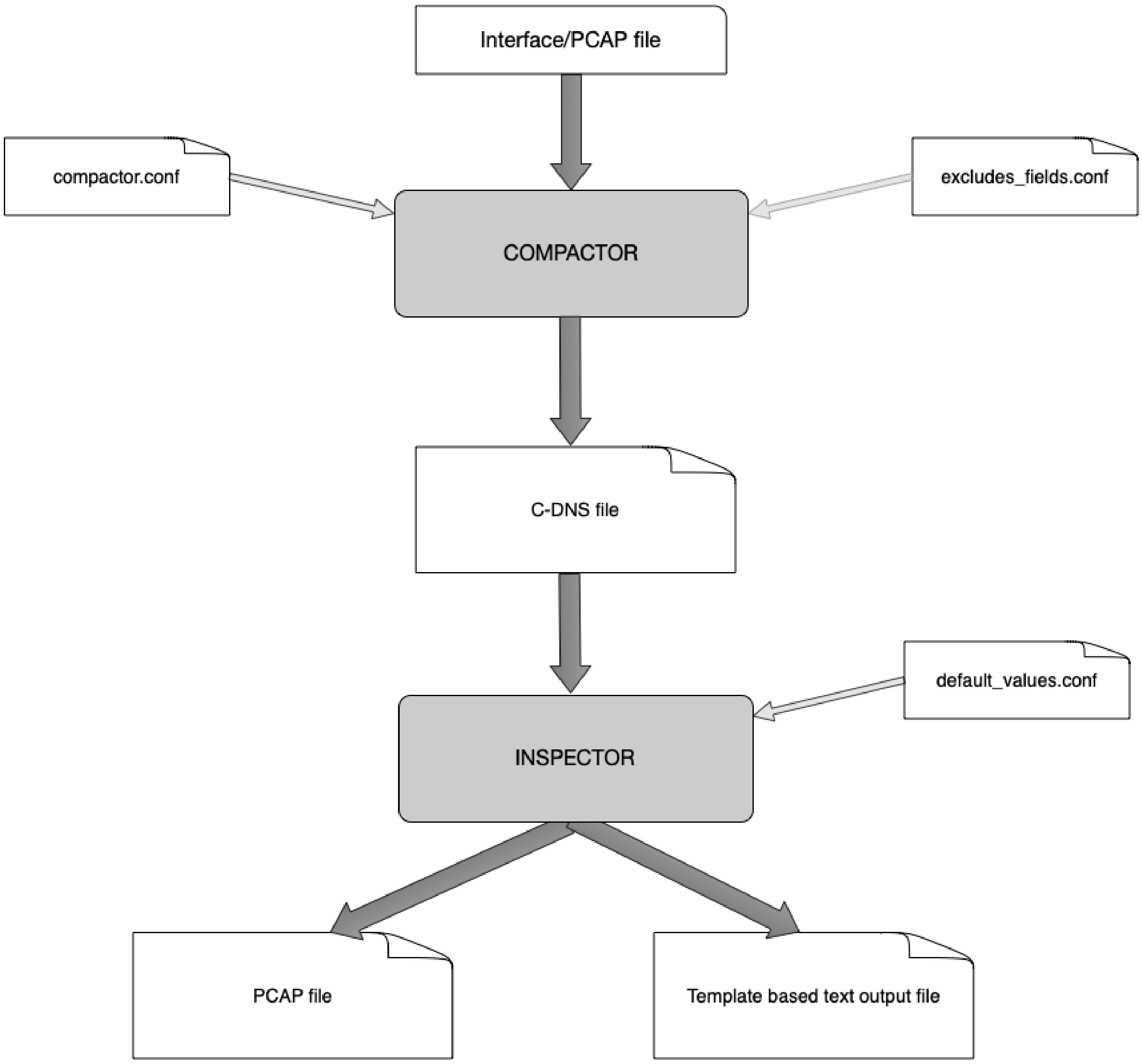
1.3.1.2. Changes in version 1.0 DNS-STATS Compactor

One significant change between the draft C-DNS specification used in earlier version of the DNS-STATS Compactor and the RFC8618 specification used in version 1.0 and later is that virtually every piece of data is now optional to capture. In contrast, the earlier versions required a fixed subset of data to always be captured with the option to include additional data. Whilst adding flexibility to the format is also adds some complexity in terms of configuration and PCAP regeneration.

Several things should be noted as a result of specification update:

- The default behaviour in terms of which data is collected will not change in 1.0 DNS-STATS Compactor, all additional sections are still collected.
- The use of **include** options to capture additional data is deprecated in 1.0 and will be removed in a future release. Users can instead use the more flexible **excluded_fields** file to better customise catpured data (see *compactor* **excluded_fields** file)
- For 1.0 C-DNS files that capture less than the minimum subset of data defined in the 0.2 and 0.5 versions, PCAP regeneration becomes more complex and requires some default values to generate sane PCAPS (see *inspector* **default_values** file).

An overview of the 1.0 processing flow is shown below.



1.3.2. General purpose compression

DNS-STATS Compactor provides options to then compress the C-DNS files using general purpose compression, producing files that are typically less than half the size of compressed raw PCAP files, while using a fraction of the CPU resources used by compressing raw PCAP.

1.4. Support

Bug reports and feature requests can be submitted via the issue tracker: https://github.com/dns-stats/compactor/issues

Known Issues are documented at https://github.com/dns-stats/compactor/blob/master/KNOWN_ISSUES.txt

A mailing list is available for users: https://mm.dns-stats.org/mailman/listinfo/dns-stats-users

2. Installation

2.1. Installing from packages

Binary install packages are available from the dns-stats Launchpad PPA (Personal Package Archive)

- `ppa:dns-stats/compactor-bionic` for Ubuntu 18.04 LTS 'Bionic Beaver'
- `ppa:dns-stats/compactor-focal` for Ubuntu 20.04 LTS 'Focal Fossa'

2.1.1. Ubuntu packages

compactor and *inspector* are supplied in separate packages named `dns-stats-compactor` and `dns-stats-inspector`.

2.1.1.1. Pre-requisites

Both *compactor* and *inspector* use libtins for various network related functions. It will be installed as a pre-requisite package automatically.

2.1.1.2. Installing

You need first to add the DNS-STATS PPA to your system's Software Sources:

```
sudo add-apt-repository ppa:dns-stats/compactor-<ubuntu_release>
sudo apt update
```

You can then install either both of the `dns-stats-compactor` or `dns-stats-inspector` packages. Their pre-requisite packages will be downloaded and installed automatically.

```
$ sudo apt install dns-stats-compactor
$ sudo apt install dns-stats-inspector
```

2.1.1.3. Post-installation

After installation *compactor* can be used from the command line, for example for ad-hoc captures or converting PCAP capture files.

Whenever the *compactor* package is installed, the system will attempt to start *compactor* as a service.

The first time the *compactor* package is installed, it installs a default configuration file `/usr/local/etc/dns-stats-compactor/compactor.conf`. This default configuration does not specify a capture interface, so the attempt at starting *compactor* as a service will fail. Before using *compactor* as a service, you need to edit the default configuration and at minimum specify a capture interface.

When the *compactor* package is upgraded, any running service is stopped for the upgrade and restarted immediately the upgrade is complete.

2.2. Installing from source

The source code is available at: https://github.com/dns-stats/compactor

Release tarballs are also available on github.

2.2.1. Pre-requisites

To build *compactor* and *inspector* from source, the following items are required. They should be available via standard installation repositories for most platforms. If not, for information on building pre-requisite items from source, see the documentation for those items.

`C++ compiler`	*compactor* and *inspector* are written in C++. Building them requires a C++ compiler and tool chain compatible with the 2011 ISO standard, otherwise C++11.
`pkg-config`	Used for dependency management.

`boost-log`
`boost-program-options`
`boost-system`
`boost-thread`
`boost-iostreams`
`boost-filesystem`
	Several libraries from Boost C++ are required. Depending on your system, it may be possible to install just the build requirements for individual libraries, or it may be more convenient to install all the Boost libraries. NOTE: *compactor* and *inspector* require Boost 1.54 or later.

`liblzma`	The compression library from XZ utils.
`libpcap`	Library for capturing network traffic http://www.tcpdump.org/.

`libtcmalloc-minimal4`	Optionally, on systems such as Linux where it’s available, `tcmalloc` from the Google performance tools gives a notable performance boost over standard `glibc` `malloc`. If not present, the build will use the standard system `malloc`.
`libtins`	Networking functions library. http://libtins.github.io/. See Pre-requisites if there is no package for your OS.
`openssl`	Cryptography library, used in pseudo-anonymisation. http://www.openssl.org/.
`libctemplate`	Text templating system. https://github.com/OlafvdSpek/ctemplate.
`libmaxminddb`	MaxMind GeoIP reader. https://github.com/maxmind/libmaxminddb.
`protobuf`	Google Protocol Buffers library. https://developers.google.com/protocol-buffers.

2.2.2. Optionally building documentation

The documentation is built using Asciidoctor, version 1.5.0 or later is required (this may require installation using 'gem install asciidoctor' on some platforms).

Doxygen documentation is also built if Doxygen is installed.

2.2.3. Building and installing

2.2.4. Building from a git repository

To build *compactor* and *inspector*, select the desired branch; for example the most recent release branch/tag or the latest development code which is in 'develop'.

```
$ git checkout <branch or tag>
$ git submodule update --init
```

The code uses GNU Autotools. Building and installing requires configuring for the locally installed version of Autotool, and then follows the usual Autotools process.

```
$ ./autogen.sh
$ ./configure
$ make
$ make install
```

As usual with Autotools, by default the install is to directories under `/usr/local`.

2.2.5. Building from a release tarball

To build *compactor* and *inspector*, unpack the release tarball.

```
$ tar -xvzf dns-stats-compactor-<version>.tar.gz
```

The code uses GNU Autotools. Building and installing follows the usual Autotools process.

```
$ cd dns-stats-compactor-<version>
$ ./configure
$ make
$ make install
```

As usual with Autotools, by default the install is to directories under `/usr/local`.

3. Configuring

Many *compactor* settings for capturing and storing traffic can be set using the *compactor* configuration file. Additionally, binary package installations of *compactor* install support for running the compactor as a daemon, and these typically restrict the system resources available to *compactor* .

inspector configuration is discussed in the Running *inspector* section.

3.1. *compactor* Command options

Command options are given on the command line.

-h, --help

Print a usage message briefly summarising these command-line options and then exit.

-v, --version

Print the version number of **compactor** to the standard output stream and then exit.

-r, --report-info [*arg*]

Report info (config and statistics summary) on exit. *arg* may be `true` or `1` to enable promiscuous mode, `false` or `0` to disable promiscuous mode. If *arg* is omitted, it defaults to `true`.

-D, --relaxed-mode [*arg*]

Warn (instead of error) if unrecognized command line or config file options are found. *arg* may be `true` or `1` to enable promiscuous mode, `false` or `0` to disable promiscuous mode. If *arg* is omitted, it defaults to `true`.

-l, --list-interfaces

List all network interfaces from which DNS traffic may be captured.

-c, --configfile [*arg*]

Read configuration from file *arg*. If not specified, the default configuration file `/usr/local/etc/dns-stats-compactor/compactor.conf` is read if present.

--excludesfile [*arg*]

Read excludes from file *arg*. If not specified, the default configuration file `/usr/local/etc/dns-stats-compactor/excluded_fields.conf` is read if present.

--debug-dns

Print a summary of each DNS packet to standard output after decoding.

--debug-qr

Print a summary of each query/response pair to standard output after matching query and response.

3.2. *compactor* configuration file

3.2.1. Configuration file location

On startup, *compactor* attempts to read a configuration file. By default this is named `compactor.conf`, and is located in a `dns-stats-compactor` system configuration directory.

If installed from a binary package on Linux, the configuration file will be at `/usr/local/etc/dns-stats-compactor/compactor.conf`.

An alternate path for the configuration file can be specified to the compactor using the **-c, --configfile** command line option.

3.2.1.1. Default configuration

Both installing via binary packages and installing using the source distribution will install a `compactor.conf` configuration file. This needs to be edited to specify an interface to collect from, but is otherwise configured to collect traffic:

- Recording all DNS additional sections
- Writing C-DNS output compressed with `xz` compression
- Naming output files named with the date, time, rotating period and capture interface.
- A new output file is commenced every 5 minutes.
- The 'raw' traffic is also recorded to a PCAP output file
- The 'ignored' traffic is also recorded to a PCAP output file
- The output files are stored under `/usr/local/var/lib/dns-stats-compactor/dns-stats-compactor` in subdirectories `cdns`, `pcap/raw` and `pcap/ignored`.

All configuration items can also be specified on the command line.

If an option appears in the configuration file and on the command line, the command line setting overrides the configuration file.

On the command line, configuration options may be specified in short and long forms. In a configuration file, the long form of the option name must be used. So, for example, to set the snap length to 64, the configuration file entry must read `snaplen=64`. `s=64` will not be accepted.

3.2.2. Configuration file format

A configuration file is a plain text file with lines in the form `option=value`. A `#` character introduces a comment that spans until the end of the line. To illustrate, here is the default configuration file:

```
#
# Configuration for DNS-STATS compactor.
#
# Commented out items show the default values.
#

# Network capture options.

# Interface on which to capture traffic.
# interface=

# VLAN IDs to capture. If none specified, all are captured.
# vlan-id=

# DNS port - only traffic to or from this port is captured.
# dns-port=53

# Snap length - limit of bytes in package to capture.
# snaplen=65535

# Filter expression.
# filter=

# Enable promiscuous mode.
# promiscuous-mode=false

# DNSTAP capture options.

# Unix socket to create for traffic capture.
# dnstap-socket=

# Attempt to set owner of Unix socket to this user.
# dnstap-socket-owner=

# Attempt to set group of Unix socket to this group.
# dnstap-socket-group=
# Attempt to set write permission of Unix socket to user only ('user'),
# user+group ('group') or all users ('all').
# dnstap-socket-write=

# Other capture options.

# Optional hints about the server addresses. These do not affect collection,
# but are stored in the C-DNS output and may be useful for downstream
# analysis. If specified, and capture is from one or more interfaces,
# the address must be present on a capture interface.
# server-address-hint=

# Omit records of this RR type from the content of DNS messages when writing output
# (note - the first question is always written). Type name in UPPER CASE e.g. AAAA,
RRSIG.
# ignore-rr-type=

# Include records of this RR type in the content of DNS messages when writing output
# (note - the first question is always written). Type name in UPPER CASE e.g. AAAA,
RRSIG.
# accept-rr-type=

# Log basic collection stats to syslog every n seconds. 0 (default) == never.
# log-network-stats-period=0

# Log detailed file processing for debugging.
# log-file-handling=false

# (Sampling is an experimental feature)
# Sampling threshold is percentage of traffic dropped above which sampling will be
enabled. Default is 10.
# sampling-threshold=10

# Sampling rate (1 in n) to be applied if packets dropped internally. 0 (default) ==
none.
# sampling-rate=0

# Apply sampling for n seconds for before re-checking for dropped packets. Default is
100.
# sampling-time=100

# Output options.

# Output file rotation period, in seconds.
# rotation-period=300

# Maximum Query/Response records in a file block.
# max-block-qr-items=5000

# Maximum size of uncompressed output before rotation triggered. 0=no limit.
# Multiplicative suffices:
#     k (1024), K (1000)
#     m (1024*k), M (1000*K)
#     g (1024*m), G (1000*M)
#     t (1024*g), T (1000*G)
# max-output-size=0

# C-DNS output file pattern.
# output=
output=/usr/local/var/lib/dns-stats-compactor/cdns/%Y%m%d-%H%M%S_%{rotate-
period}_%{interface}.cdns

# Raw PCAP output file pattern.
# raw-pcap=
raw-pcap=/usr/local/var/lib/dns-stats-compactor/pcap/raw/%Y%m%d-%H%M%S_%{rotate-
period}_%{interface}.raw.pcap

# Ignored PCAP output file pattern.
# ignored-pcap=
ignored-pcap=/usr/local/var/lib/dns-stats-compactor/pcap/ignored/%Y%m%d-
%H%M%S_%{rotate-period}_%{interface}.ignored.pcap

# The use of include lines is deprecated. See the user_guide for more information.
# Specify which optional sections to capture. Value may be one of:
# query-questions, query-answers, query-authority, query-additional,
# response-questions, response-answers, response-authority, response-additional,
# query-all, response-all, all.
#
# The line may be repeated to request multiple optional sections, for example:
# include=query-questions
# include=response-answers
#
# If no include is specified then no optional sections are captured.
include=all

# maximum number of compression threads.
# max-compression-threads=2

# Compress C-DNS using gzip?
# gzip-output=false

# C-DNS gzip compression level.
# gzip-level=6

# Compress C-DNS using xz?
xz-output=true

# C-DNS xz compression level.
# xz-preset=6

# Compress PCAP using gzip?
# gzip-pcap=false

# PCAP gzip compression level.
# gzip-level-pcap=6

# Compress PCAP using xz?
# xz-pcap=false

# PCAP xz compression level.
# xz-preset-pcap=6

# Query matching options.

# Seconds to wait for response before timing out query.
# query-timeout=5

# Microseconds to wait for query arrive after response received.
# skew-timeout=10
```

3.2.3. Configuration options

3.2.3.1. Capture from network

-i, --interface *arg*

Capture traffic from network interface *arg*. This argument may be given multiple times to allow capture for several interfaces in parallel.

-S, --server-address-hint *arg*

arg is a single IPv4 or IPv6 address for the server. These hints are optional, do not affect capture in any way, but are stored in C-DNS as a potential aide to post-capture analysis. If capture from one or more interfaces is specified, then a sanity check is performed to ensure the address is be present on a system interface; if not, an error is logged, but collection proceeds. This argument may be given multiple times.

-f, --filter *arg*

Discard packets not matching the filter expression *arg*. The filter syntax is described in **pcap-filter**(7). Packets are discarded at the point of capture; discarded packets do not appear in any ignored PCAP output file.

--dns-port *arg*

Traffic to or from port *arg* is DNS traffic and should be recorded. Traffic to other ports is ignored. The default is 53.

-s, --snaplen *arg*

Capture up to *arg* bytes per packet. The default is 65535.

-p, --promiscuous-mode [*arg*]

Put the interface into promiscuous mode. *arg* may be `true` or `1` to enable promiscuous mode, `false` or `0` to disable promiscuous mode. If *arg* is omitted, it defaults to `true`. Promiscuous mode is disabled by default.

-a, --vlan-id *arg*

ID of VLAN to be captured if on a 802.1Q network. The argument may be given multiple times to capture from several VLANs. If no **vlan-id** argument is given, traffic is captured from all VLANs.

-L, --log-network-stats-period *arg*

Every *arg* seconds, log basic statistics on packet collection to the system log. The default value of 0 disables this logging.

-F, --log-file-handling

Log detailed file processing when files are rotated and compressed. This facilitates debugging of file processing issues and measurement of C-DNS file compression times.

--sampling-threshold *arg*

A threshold for the percentage of traffic dropped on the internal channels above which sampling will be enabled (if **sampling-rate** is greater than 0). The default value is 10.

--sampling-rate *arg*

The rate (1 in *arg* packets) to be applied when sampling mode is enabled (this is an experimental feature). The rate is applied for **sampling-time** seconds and then sampling is disabled. After this, depending on the traffic rate, sampling may be enabled again if the drops rise above the **sampling-threshold**. The default value of 0 disables this option.

--sampling-time *arg*

The period of time to apply sampling mode for. To avoid accidentally setting a low value that could result in instability this must be at least 10s. The default value is 100.

3.2.3.2. Capture from DNSTAP

If **compactor** has been built with DNSTAP enabled, the following options control capture from DNSTAP.

--dnstap [*arg*]

When reading input data from file, treat the data as DNSTAP data not PCAP data if *arg* is `true` or `1`. The default is `false`.

--dnstap-socket *arg*

Capture DNSTAP traffic from Unix socket *arg*.

--dnstap-socket-owner *arg*

If specified, **compactor** will try to set the owner of the DNSTAP socket to *arg*.

--dnstap-socket-group *arg*

If specified, **compactor** will try to set the group of the DNSTAP socket to *arg*.

--dnstap-socket-write ***arg***

If specified, **compactor** will try to set write permissions on the DNSTAP socket. If *arg* is **owner**, the socket owner will have write permission. If *arg* is **group**, the socket owner and group will have write permission. If *arg* is **all**, all users will have write permission. No other values of *arg* are permitted.

3.2.3.3. Outputs

-o, --output ***PATTERN***

Use *PATTERN* as the template for the file path for the C-DNS output files. If no output pattern is given, no output is written.

-z, --gzip-output [*arg*]

Compress data in the C-DNS output files using gzip(1) format. *arg* may be `true` or `1` to enable compression, `false` or `0` to disable compression. If *arg* is omitted, it defaults to `true`. If compression is enabled, an extension `.gz` is added to the output filename.

-y, --gzip-level [*arg*]

Compression level to use when producing gzip(1) C-DNS output. *arg* must be a single digit `0` to `9`. If not specified, the default level is `6`.

-x, --xz-output [*arg*]

Compress data in the C-DNS output files using xz(1) format. *arg* may be `true` or `1` to enable compression, `false` or `0` to disable compression. If *arg* is omitted, it defaults to `true`. If compression is enabled, an extension `.xz` is added to the output filename.

-u, --xz-preset [*arg*]

Compression preset level to use when producing xz(1) C-DNS output. *arg* must be a single digit `0` to `9`. If not specified, the default level is `6`.

--max-compression-threads [*arg*]

Maximum number of threads to use when compressing. Compression uses one thread per output file, so this argument gives the number of output files that can be compressed simultaneously. *arg* must be `1` or more. If not specified, the default number of threads is `2`.

-w, --raw-pcap ***PATTERN***

Use *PATTERN* as the template for a file path for output of all packets captured via network capture to file in PCAP format. If no pattern is given, no raw packet output is written. This option is currently ignored when capturing via DNSTAP.

-m, --ignored-pcap ***PATTERN***

Use *PATTERN* as the template for a file path for output of all packets captured vai network capture that were not to the configured DNS ports, or were not validly formed DNS packets. If no pattern is given, no ignored packet output is written. This option is currently ignored when capturing via DNSTAP.

-Z, --gzip-pcap [*arg*]

Compress data in the PCAP output files using gzip(1) format. *arg* may be `true` or `1` to enable compression, `false` or `0` to disable compression. If *arg* is omitted, it defaults to `true`. If compression is enabled, an extension `.gz` is added to the output filename.

-Y, --gzip-level-pcap [*arg*]

Compression level to use when producing gzip(1) PCAP output. *arg* must be a single digit `0` to `9`. If not specified, the default level is `6`.

-X, --xz-pcap [*arg*]

Compress data in the PCAP output files using xz(1) format. *arg* may be `true` or `1` to enable compression, `false` or `0` to disable compression. If *arg* is omitted, it defaults to `true`. If compression is enabled, an extension `.xz` is added to the output filename.

-U, --xz-preset-pcap [*arg*]

Compression preset level to use when producing xz(1) C-DNS output. *arg* must be a single digit `0` to `9`. If not specified, the default level is `6`.

-t, --rotation-period *SECONDS*

Specify the frequency with which all output file path patterns should be re-examined. If the file path has changed after this period (e.g. because it contains a date/time element which has changed), the existing output file is closed and a new one opened using the new pattern expansion. If the file path has not changed, the pattern is re-examined every second until it changes. Note that file path patterns that do not contain date/time elements will therefore not trigger file rotation via this mechanism, so using a default file pattern similar that that in the default configuration file is recommended. The default period is 300 seconds. This may be combined with maximum output file size rotation, in which case rotation happens when either condition is met.

-n, --include *SECTIONS*

Indicate which optional sections should be included in the main output. This argument can be given multiple times. By default none of these optional sections are included. This option is deprecated. and will be removed in a future release. It is replaced by the `excludesfile` option. See the user guide for more information.

-G, --ignore-opcode *OPCODE*

DNS messages with this *OPCODE* should NOT be included in the main output (the DNS message is discarded). This argument can be given multiple times.

-E, --accept-opcode *OPCODE*

DNS messages with this *OPCODE* should be included in the main output (if used, messages with OPCODES not specified with this option are discarded). This argument can be given multiple times.

-g, --ignore-rr-type *TYPE*

Records of this RR type *TYPE* should NOT be included in DNS messages when they are written to the main output. Note that the first question is ALWAYS included, but *TYPE* nominated using the RR type name (in upper case) will be omitted from any other question and from any Answer, Authority or Additional section. This argument can be given multiple times.

-e, --accept-rr-type *TYPE*

Records of this RR type *TYPE* should be included in DNS messages when they are written to the main output (if used, RR types not specified with this option will be omitted from DNS messages). Note that the first question is ALWAYS included, but only *TYPE* nominated using the RR type name (in upper case) will be included in any other question and in any Answer, Authority or Additional section. This argument can be given multiple times.

--max-block-items *arg*

Set the maximum number of query/response items or address event items included in a single output C-DNS block. *arg* must be a positive integer. The default maximum size is 5000.

--max-output-size *arg*

Sets a maximum size for the uncompressed output before an output file rotation is triggered. *arg* must be a positive integer, and may optionally be followed by one of the following multiplicative suffixes: *k*=1024, *K*=1000, *m*=1024*1024, *M*=1000*1000 and similarly for *g*, and *t*. If a file rotation is triggered, the remaining block and the file postlude will be written, so the final file size will exceed this setting by a small margin. The default value is 0, which indicates there is no maximum size. This may be combined with a rotation period, in which case rotation happens when either condition is met.

--client-address-prefix-ipv4 *arg*

Set the prefix size (number of address bits stored) for IPv4 client addresses. The client address is the address of the sender of a query or the receipient of a response. *arg* must be a positive integer less than or equal to 32. The default is 32, so the entire IPv4 address is stored.

--client-address-prefix-ipv6 *arg*

Set the prefix size (number of address bits stored) for IPv6 client addresses. The client address is the address of the sender of a query or the receipient of a response. *arg* must be a positive integer less than or equal to 128. The default is 128, so the entire IPv6 address is stored.

--server-address-prefix-ipv4 *arg*

Set the prefix size (number of address bits stored) for IPv4 server addresses. The server address is the address of the recipient of a query or the sender of a response. *arg* must be a positive integer less than or equal to 32. The default is 32, so the entire IPv4 address is stored.

--server-address-prefix-ipv6 *arg*

Set the prefix size (number of address bits stored) for IPv6 server addresses. The server address is the address of the recipient of a query or the sender of a response. *arg* must be a positive integer less than or equal to 128. The default is 128, so the entire IPv6 address is stored.

3.2.3.4. Query/response matching

-q, --query-timeout *SECONDS*

If no response is found for a query after *SECONDS*, time out the query. The default timeout is 5 seconds. Fractional values of *SECONDS* down to milliseconds may be given.

-k, --skew-timeout *MICROSECONDS*

Due to the vagaries of the network stack, it is possible for responses to be reported before the matching query, even though the query has an earlier timestamp than the response. A response is not considered to be missing a query until after *MICROSECONDS*. The default timeout is 10 microseconds.

3.2.3.5. Output file patterns

Output files, C-DNS and PCAP, are named using output file patterns. These are made up from a directory path and an output filename. So, for example, a PCAP output file might be named `/tmp/pcap/output.pcap`. In this example, `/tmp/pcap/` is the directory path, and `output.pcap` is the output filename.

Output filenames can contain expansion patterns. Expansion patterns are introduced by a `%` character, and are of two basic types, time expansions and configuration expansions.

`%` followed by a single letter	Insert a time expansion.
`%{name}`	Insert the current value of the configuration item `name`
`%%`	Insert a single `%` in the output filename.

The full list of letters available in the time expansion, and what they expand to, is in the **strftime**(3) manual page. Some of the commonly used ones are given below, with expansions for a date and time of Monday January 16th 2017 at 13:18:05.

`%y`	The year, not including the century.	`17`
`%Y`	The year, including the century.	`2017`
`%C`	The century part of the year.	`20`
`%m`	The month as a decimal number (01-12).	`1`
`%d`	The day of the month as a decimal number (01-31).	`16`
`%w`	The day of the week as a decimal number (0-6). Sunday is 0.	`1`
`%W`	The week number (00-53).	`03`
`%H`	The hour (24-hour clock) as a decimal number (00-23).	`13`
`%M`	The minute as a decimal number (00-59).	`18`
`%S`	The second as a decimal number (00-59).	`05`

Not all configuration items can be used in a configuration expansion. The items that can be used are as follows.

`%{interface1}`	The name of the first configured interface. `interface2` gives the second interface, `interface3` the third and so on.	`eth0`
`%{interface}`	The names of all configured network interfaces separated by `-`, or `dnstap` if capturing with DNSTAP.	`eth0-eth1`

`%{rotate-period}`	The file rotation period used when file names contain time/date elements, in seconds.	`300`
`%{snaplen}`	The network capture snap length.	`65535`
`%{query-timeout}`	The query timeout, in seconds. If no response to a query arrives by the timeout, the query is treated as unanswered.	`5`
`%{skew-timeout}`	The skew timeout, in microseconds. If a response arrives without a query, it is held for the timeout period to see if a query arriving just after matches.	`10`
`%{promiscuous-mode}`	Outputs `1` if the network interfaces are in promiscuous mode, `0` otherwise.	`true`
`%{vlan-id1}`	The ID of the first configured VLAN. `vlan-id2` gives the second configured VLAN ID, `vlan-id3` the third and so on.	`eth0`
`%{vlan-id}`	The IDs of all configured VLANs separated by `-`.	`10-12`

Example:

output=*PATTERN*

Use *PATTERN* as the template for the file path for the C-DNS output files. If no output pattern is given, no output is written.

```
output=/tmp/cdns/%Y%m%d-%H%M%S_%{rotate_period}_%{interface}.cdns
```

Using the above date and time, a rotation period of 300s and collecting from interfaces `eth0` and `eth1` this will write to `/tmp/cdns/20170116-131805_300_eth0-eth1.cdns`.

3.2.3.6. C-DNS options

include=*SECTIONS*

Indicate which optional sections should be included in the C-DNS output. This argument can be given multiple times. If no include is specified then none of these optional sections are included.

The use of **include** lines is deprecated in v1.0 and will be removed in a future major release. The functionality to control what data is captured has moved to the new **excluded_fields** file. See *compactor* **excluded_fields** file.

Section name	Description
query-questions	Include second and subsequent QUESTION sections from queries. The first QUESTION section is always recorded.
query-answers	Include ANSWERS data from queries.
query-authority	Include AUTHORITY data from queries.
query-additional	Include ADDITIONAL data from queries.

Section name	Description
query-all	Include all sections from queries.
response-questions	Include second and subsequent QUESTION sections from responses. The first QUESTION section is always recorded.
response-answers	Include ANSWERS data from responses.
response-authority	Include AUTHORITY data from responses.
response-additional	Include ADDITIONAL data from responses.
response-all	Include all sections from queries.
all	Include all sections from both queries and responses.

```
include=all
```

3.3. *compactor* excluded_fields file

The **excluded_fields** file replaces the deprecated **include** functionality currently found in the configuration file. By default in v1.0 this file is not used, the deprecated **include** functionality is still used for backwards compatibilty.

Users must use either the deprecated **include** configuration options or the new **excluded_fields** file. They cannot be combined. If the **excluded_fields** file exists AND there are **include** lines in the configuration file (or provided on the command line) an error will be logged and *compactor* will exit.

If the **excluded_fields** file does exists the *compactor* will act as if **include=all** was present. The **excluded_fields** file then controls what fields are **excluded** from the C-DNS output. Uncommenting a field leads to it being omitted from the capture.

The **excluded_fields** are stored in the C-DNS file as 'StoreageHints' for *inspector* to use when processing C-DNS file, see Sections 6.2.1 and 7.3.1.1.1.1 in RFC8618 for more details.

3.3.1. Excluded_fields file location

On startup, *compactor* looks for a **excluded_fields** file named `excluded_fields.conf` located in a `dns-stats-compactor` system configuration directory. A different file name and location can specified on the command line by using the *compactor --excludesfile* option.

3.3.2. Default configuration

By default only a sample **excluded_fields** file is provided. If installed from a binary package on Linux, the file will be at `/usr/local/etc/dns-stats-compactor/excluded_fields.conf.sample`.

This file (with the .sample extension removed) can be used to specify which fields are to be excluded from the C-DNS capture (assuming the **include** option is not used). In the sample all the fields are commented out so all fields would be collected in the C-DNS capture, which gives the same behaviour as **include**=**all**.

The fields are specified in logical groups.

```
# A list of fields to be omitted from compactor outputs.
# Uncomment the field for it to be omitted.
# The fields MUST go under the heading as shown

[ip-header]
# time-offset
# response-delay
# client-address
# client-port
# client-hoplimit
# server-address
# server-port
# qr-transport-flags
# qr-type

[dns-header]
# transaction-id
# query-opcode
# query-rcode
# dns-flags
# response-rcode
# query-qdcount
# query-ancount
# query-arcount
# query-nscount

[dns-payload]
# query-name
# query-class-type
# rr-ttl
# rr-rdata
# query-udp-size
# query-opt-rdata
# query-edns-version
# query-question-sections
# query-answer-sections
# query-authority-sections
# query-additional-sections
# response-answer-sections
# response-authority-sections
# response-additional-sections

[dns-meta-data]
# query-size
# response-size
# qr-sig-flags

[storage-meta-data]
# address-events
```

3.3.3. Other configurations

As previously noted, excluding some data could have implications for PCAP regeneration, see *inspector* **default_values** file. The following is a suggested **excludes_fields** file which results in the smallest capture files where the goal is to be able to reasonably reconstruct just queries in a PCAP (when using the *inspector* with the -q option). Capturing less data will require populating the PCAP with default values for some fields.

```
[ip-header]
# time-offset
response-delay
# client-address
# client-port
# client-hoplimit
# server-address
# server-port
# qr-transport-flags

[dns-header]
# transaction-id
# query-opcode
# query-rcode
# dns-flags
response-rcode
query-qdcount
query-ancount
query-arcount
query-nscount

[dns-payload]
# query-name
# query-class-type
# rr-ttl
# rr-rdata
# query-udp-size
# query-opt-rdata
# query-edns-version
# query-question-sections
# query-answer-sections
# query-authority-sections
# query-additional-sections
response-answer-sections
response-authority-sections
response-additional-sections

[dns-meta-data]
query-size
response-size
qr-sig-flags

[storage-meta-data]
address-events
```

A more detailed discussion of the issues around C-DNS to PCAP reconstruction can be found in Section 6.2.1, Section 9 and Appendix D of RFC8618.

3.4. Configuring *compactor* daemon startup

All binary packages of *compactor* include startup setup allowing *compactor* to be run as a daemon, and possibly started automatically on boot.

These startup setups may also contain settings constraining the compactor’s use of memory and CPU.

3.4.1. Linux with `systemd`

By default, Ubuntu 16.04 LTS 'Xenial Xerus' and later releases use `systemd`.

3.4.1.1. Running as a daemon

Binary packages for Ubuntu include a `systemd` service file with the setup information required to run *compactor* as a daemon.

When installing on Debian-based systems such as Ubuntu, installing the package will automatically enable the service and attempt to start *compactor* , or restart it if already running.

To enable the service, use the `systemctl enable` subcommand.

```
# systemctl enable dns-stats-compactor
```

To start or stop the daemon, or request it reload its configuration, use the appropriate `systemctl` subcommand.

3.4.1.2. Changing resource restrictions

This file includes `CPUAffinity` and `MemoryLimit` clauses to restrict *compactor* to particular CPUs and limit its memory usage. In the installed service file, these are set to CPU 0 only and 1Gb respectively.

```
[Service]
CPUAffinity=0
MemoryLimit=1G
```

To override these, use the `systemctl edit` subcommand to create a service file override unit with an updated version of the above snippet.

4. Running *compactor*

4.1. Stopping and starting *compactor* daemon

If *compactor* was installed from a binary package, then once the configuration has been set up *compactor* can be run as a service or daemon.

4.1.1. Linux with `systemd`

By default, Ubuntu 16.04 LTS 'Xenial Xerus' and later releases use `systemd`.

compactor service is a standard `systemd` service. It can be manually started, stopped and requested to reload its configuration.

```
# systemctl start dns-stats-compactor
# systemctl stop dns-stats-compactor
# systemctl reload dns-stats-compactor
```

Stopping the service is equivalent to manually interrupting a capture. See Capturing network traffic.

Requesting the service reloads its configuration is equivalent to a manual request for restarting capture. See Restarting capture with modified configuration.

4.2. Running *compactor* from the command line

compactor can be run from the command line to either:

- perform ad-hoc captures from one or more network interfaces, or
- perform ad-hoc captures from DNSTAP, or
- convert already captured PCAP files.

Full details of available command options are in *compactor* manual page, **compactor**(1). A summary is available:

```
$ compactor  -h
Usage: compactor  [options] [capture-file ...]
Options:

Command options:
  -h [ --help ]                     show this help message.
  -v [ --version ]                  show version information.
  -c [ --configfile ] arg           configuration file.
  --excludesfile arg                exclude hints file.
  -r [ --report-info ] [=arg(=1)]   report info (config and stats summary) on
                                    exit.
  -D [ --relaxed-mode ] [=arg(=1)] parse command line allowing
                                    unrecognized options but warning.
  --debug-dns [=arg(=1)]            print DNS packet details.
  --debug-qr [=arg(=1)]             print Query/Response match details.
  -l [ --list-interfaces ]          list all network interfaces.

Configuration:
  -t [ --rotation-period ] arg (=300)   rotation period for filename based rotation
                                        for all outputs, in seconds.
  -q [ --query-timeout ] arg (=5)       timeout period for unanswered queries,
                                        in seconds.
...(rest of output omitted for brevity)...
```

Specifying a combination of capture files, network interfaces or DNSTAP capture on the command line is an error.

Options `--debug-dns` and `--debug-qr` output human-readable summary packet and match data as *compactor* is running. They are a useful way to verify operation.

4.2.1. Capturing network traffic

If at least one network interface is specified and *compactor* has permission to read from the network, *compactor* will start recording network traffic.

For example, to capture traffic on `eth0` to output `capture.cdns`, capturing all DNS sections and automatically `xz` compressing the output:

```
$ compactor -x -o capture.cdns -n all -i eth0
```

compactor will capture traffic until interrupted by a signal, for example the user typing Control-C. If the -r option is used then *compactor* will report summary information on exit (configuration and basic statistics).

If C-DNS compression is enabled, interrupting *compactor* causes the current compression to be aborted (the '.raw' uncompressed file is retained on disk). For this reason you are recommended to have C-DNS output file rotation enabled when using compression. For more details, see C-DNS output compression.

4.2.2. Capturing DNSTAP traffic

If a DNSTAP Unix socket is specified, and *compactor* has permission to create the socket, and a local nameserver is configured to write DNSTAP to that socket, *compactor* will start recording network traffic via DNSTAP.

For example, to capture traffic on `/run/unbound-dnstap.sock` to output `capture.cdns`, capturing all DNS sections and automatically `xz` compressing the output:

```
$ compactor -x -o capture.cdns -n all --dnstap-socket /run/unbound-dnstap.sock
```

compactor will capture traffic until interrupted by a signal, for example the user typing Control-C. If the -r option is used then *compactor* will report summary information on exit (configuration and basic statistics).

If C-DNS compression is enabled, interrupting *compactor* causes the current compression to be aborted (the '.raw' uncompressed file is retained on disk). For this reason you are recommended to have C-DNS output file rotation enabled when using compression. For more details, see C-DNS output compression.

When capturing traffic over DNSTAP, output of raw or ignored packets to PCAP is not currently supported. Any related *compactor* options specified will be ignored.

4.2.3. Capturing from PCAP files

Unless DNSTAP input files are specified, any non-option parameters on the command line are assumed to be PCAP files to be used for input to *compactor* . In this case, any capture interface specified in the configuration file is ignored and the PCAP files used as input.

So, to convert input PCAP file `capture.pcap` to output `capture.cdns`, capturing all DNS sections and automatically `xz` compressing the output:

```
$ compactor -x -o capture.cdns -n all capture.pcap
```

or, using the longer option forms,

```
$ compactor --xz-output --output capture.cdns --include all capture.pcap
```

Some command line options take an optional argument. For example, `--xz-output` can optionally be followed by an explicit value; if none is present then `true` is assumed. This does mean that if one of these options is the last option before the PCAP file arguments, the compactor can’t tell that the PCAP file argument is not a (probably incorrect) value for the option.

```
$ compactor -o capture.cdns -x capture.pcap
Error: the argument ('capture.pcap') for option '--output' is invalid. Valid choices
are 'on|off', 'yes|no', '1|0' and 'true|false'
```

In this case, the option argument must be given explicitly.

```
$ compactor -o capture.cdns -x true capture.pcap
```

4.2.4. Capturing from DNSTAP files

If DNSTAP input is specified by using the `-T`/`--dnstap` option, any non-option parameters on the command line are assumed to be DNSTAP files to be used for input to *compactor* . In this case, any capture interface specified in the configuration file is ignored and the DNSTAP files used as input.

So, to convert input DNSTAP file `capture.dnstap` to output `capture.cdns`, capturing all DNS sections and automatically `xz` compressing the output:

```
$ compactor -T -x -o capture.cdns -n all capture.dnstap
```

or, using the longer option forms,

```
$ compactor --dnstap --xz-output --output capture.cdns --include all capture.dnstap
```

As with other command line options, `--dnstap` can optionally be followed by an explicit value; if none is present then `true` is assumed. This does mean that if it is the last option before the DNSTAP file arguments, the compactor can’t tell that the DNSTAP file argument is not a (probably incorrect) value for the option.

```
$ compactor -o capture.cdns -x -T capture.dnstap
Error: the argument ('capture.dnstap') for option '--output' is invalid. Valid choices
are 'on|off', 'yes|no', '1|0' and 'true|false'
```

In this case, the `--dnstap` argument must be given explicitly.

```
$ compactor -o capture.cdns -x -T true capture.dnstap
```

4.2.5. Network capture permissions

If performing an ad-hoc network capture, *compactor* must have the necessary permissions to allow it to capture traffic from the specified network ports. These permissions are operating system dependent, and a full discussion is beyond the scope of this guide. The documentation for the `wireshark` tool provides a good introduction when discussing permissions for `dumpcap`. For the impatient who have the necessary permission, just run *compactor* as `root`.

4.2.6. DNSTAP capture permissions

When capturing DNSTAP over a socket, *compactor* is responsible for the creation of that socket. It will also be necessary to ensure that the local nameserver has permissions allowing it to write to the socket.

As a convenience, *compactor* provides options to set the socket owner, group and write permissions.

- `--dnstap-socket-owner`. Set the socket owner to the named user.
- `--dnstap-socket-group`. Set the socket group to the named group.
- `--dnstap-socket-write`. Give write permission to owner, owner+group or all.

Obviously, *compactor* must itself have permission to make the changes.

4.2.7. Restarting capture with modified configuration

If *compactor* is performing a capture from the network or from DNSTAP, it is possible to modify settings in the configuration file and have *compactor* re-read the configuration file. To do so, send the `HUP` signal to the *compactor* process. This will stop the current capture, re-read the configuration file, and restart capture.

Any options given on the command line will still be applied for the restarted capture, and will still over-ride any configuration file value for those options.

The process of stopping the current capture, re-reading configuration and restarting capture will mean that *compactor* will miss some traffic. The amount of time taken to stop, write queued data and then restart is heavily dependent on the load on the network at the time.

If you send a `HUP` signal to *compactor* while it is doing a capture from a file rather than a network capture, the conversion is stopped immediately and *compactor* exits.

If compression of C-DNS output is enabled, a `HUP` signal to *compactor* during a capture from a PCAP file have the same effect as interrupting *compactor*, so the current C-DNS output is lost. See C-DNS output compression for details.

4.2.8. Configuration file

Remember that when *compactor* is installed, either using a binary package or using the source distribution, a configuration file is installed with sample settings. The sample file specifies output file paths, `xz` compression on C-DNS output and all sections to be captured.

Any setting given on *compactor* command line automatically over-rides any value for that setting in the configuration file.

The **-c, --configfile** command line option over-rides the default configuration file location. If not using an alternate configuration file, but specifying all the compactor options from the command line, it may be wise to explicitly give `/dev/null` as the configuration file, to ensure that no values from the installed configuration file are used.

4.3. *compactor* log messages

compactor logs some messages to the system log. There are error messages, reporting an error within *compactor* , and also some informational messages.

Log type	Message	Comment
ERROR	Dropping on these channels: Sniffer Matcher C-DNS	Packets are arriving faster than *compactor* can process them, so packets are being dropped internally on the specified channels. More information is available via the **--log-network -stats-period** logging output.
ERROR	Dropping on these channels: Ignored-PCAP	More ignored packets - that is, packets that do not appear to be DNS related or which are malformed - are arriving than can be processed and recorded to the ignored PCAP output. At least one has been dropped.
ERROR	Dropping on these channels: Raw-PCAP	More input packets are arriving than can be processed and recorded to the raw PCAP output. At least one has been dropped.
WARNING	Sampling mode switched on for 100s with rate of 1 in 10	Drops on at least one internal channel are occurring at a rate higher than specified by the **--sampling-threshold** option so sampling is enabled.
WARNING	Sampling mode extended as drops still occurring	Drops above the threshold level are still occurring after the specified **--sampling-time** period.
WARNING	Sampling mode switched off because time limit expired and not dropping above threshold	Drops are now below the sampling threshold and so sampling is disabled after the specified time limit.
WARNING	Aborting compression of <file>	On an interrupt of kill of the process, all in progress C-DNS file compression is aborted. However the '.raw' (uncompressed) file is left on disk.
INFO	Total packet count, etc.	Basic statistics on the ongoing network capture requested by the **--log-network-stats-period** option (see below).
INFO	Starting network capture	*compactor* is starting a network capture.

Log type	Message	Comment
INFO	Rotating file to <filename>	*compactor* is rotating the current collection file to the named file.
INFO	Re-reading configuration	*compactor* has received a `HUP` signal and is re-reading its configuration.
INFO	Signal handler: Received - <signal>	*compactor* has received a signal.
INFO	Collection interrupted	*compactor* has received a signal requesting termination and is stopping collection.

Other error messages are reporting an internal error.

Example statistics produced by enabling the **log-network-stats-period** option:

```
*Stats interval: average rate          1896 pps  over  1s
 LIBPCAP : recv/OS drop/IF drop        1896/        0/        0
 Sniffer : recv/dropped/queue          1896/        0/        1
 Matcher : recv/dropped/queue          1896/        0/        0
 CDNS    : recv/dropped/queue          1896/        0/        0
 CDNS out: writ/% traffic              1896/      100/
 PCAP out: raw drop/ignored drop          0/        0/
```

These statistics provide detail about the internal components of compactor that may need to drop packets under heavy load. If sampling is enabled, an additional line outputs data on sampling:

```
 Sampling: recv/discard/state           1896/        0/       OFF
```

Note that the LIBPCAP statistics provided here are information only and may not be reliable, particularly at high load.

Additional file handling messages can be output by enabling the **--log-file-handling** option. This allows detailed debugging of file processing problems and also measurements of file compression times.

4.4. *compactor* performance considerations

4.4.1. Threading

compactor is multi-threaded. Packet parsing and query/response matching happens on the main thread, while separate threads are used for packet capture, to write C-DNS, raw PCAP and ignored PCAP outputs.

If `xz` or `gzip` compression is requested for C-DNS, that compression happens in one or more further threads as described below. `xz` or `gzip` compression in PCAP outputs remains in the PCAP output thread because the PCAP format is very simple and requires minimal processing overhead. C-DNS, on the other hand, requires non-trivial processing to perform the data de-duplication before proceeding to general purpose compression.

Data is passed between threads using queues with maximum length. If the rate of incoming data overwhelms a thread and it can’t keep up, the data is discarded and the discard recorded. As data rates rise, therefore, the compactor will keep running but more and more queries will not be processed correctly.

> By default, installing *compactor* via a binary package and running it as a service restricts *compactor* to using just one physical core. Increasing the number of cores available may, depending on the configuration, increase the maximum throughput of *compactor* .

4.4.1.1. C-DNS output compression

The processed used for C-DNS compression is designed for a scenario where *compactor* is outputting data to a C-DNS file that is periodically rotated; that is, the existing output file is closed and a new output file with a distinct name is started. C-DNS output is first written uncompressed to a '.raw' temporary output file. When the file is complete, either at the end of input from a PCAP file being converted, or after a file rotation, a new thread is spawned to compress the temporary output file to the final compressed C-DNS file. If strong compression is being used, the compression may not finished before the next output file is ready for compression, so it may be necessary to have two or more compression threads executing simultaneously to keep up. The maximum number of compression threads that may be active at any time is set in configuration; if the limit is reached, C-DNS output blocks until one of the compression threads finishes.

Detailed logs of file processing can be enabled with the **--log-file-handling** option.

> If *compactor* is interrupted, e.g. by the user typing Control-C or the service being restarted, the C-DNS output is stopped and all compressing threads are requested to abort. In this case all incomplete compressed output files are deleted, but the temporary uncompressed files are retained (alongside the completed compressed files). The files will have a `.raw` extension and can be renamed and compresses manually. Since these files are uncompressed and could be large, users should choose how to manage such files.
>
> This behaviour was changed in 1.2.1. Prior to that all in progress output files were removed on interrupt.

If the *compactor* is listening on an interface where the DNS traffic is very low or stops for some reason then output files may not get rotated and/or compressed until a) in the case of the CDNS output another DNS packet is seen or b) in the case of PCAP output any packet is seen.

Alternatively a C-DNS file rotation can be forced by sending a SIGUSR1 to `compactor`, however it is recommended to either

1. rely on just the file rotation configuration parameters OR
2. to override the internally triggered rotation with an external signal sent at a period much shorter then the file-rotation-period (which defaults to 300s)

The second mechanism must be used with care to avoid a race condition because the internal triggering cannot currently be disabled if the filename contains a timestamp. The file rotation mechanism is under review and may be further updated in a coming release.

If *compactor* is requested to reload its configuration via `HUP` signal, existing compression threads are not affected.

Note that if PCAP compression is specified, PCAP files are compressed as they are written (not via a two-stage processes as above). On interrupt these files are simply closed.

5. Running *inspector*

inspector is run from the command line to convert C-DNS from one or more C-DNS file to either

- PCAP format (the default) or
- text output based on a template file specifying the output for each query/response record.

Each output file may be accompanied by a text `info` file giving basic information on the C-DNS file contents (configuration and basic statistics). Alternatively just the `info` file can be output with no output files.

`info` files are not part of the C-DNS specification. Their contents are specific to this implementation, and are subject to change.

Bulk processing can be achieved by writing a custom script to use *inspector* to process the C-DNS files as needed.

Full details of available command options are in the *inspector* manual page, **inspector**(1). A summary is available:

```
$ inspector -h
Usage: inspector [options] [cdns-file [...]]
Options:
  -h [ --help ]                         show this help message.
  -v [ --version ]                      show version information.
  --defaultsfile arg (=/usr/local/etc/dns-stats-compactor/dns-stats-
compactor/default_values.conf)
                                        default values file.
  -o [ --output ] arg                   output file name.
  -F [ --output-format ] arg            output format. 'pcap' (default) or
                                        'template'.
  -t [ --template ] arg                 name of template to use for template
                                        output.
  -V [ --value ] arg                    <key>=<value> to substitute in the
                                        template. This argument can be
                                        repeated.
  -g [ --geoip-db-dir ] arg (=/usr/local/var/lib/dns-stats-compactor/GeoIP)
                                        path of directory with the GeoIP
                                        databases.
  -z [ --gzip-output ]                  compress output data using gzip. Adds
                                        .gz extension to output file.
  -y [ --gzip-level ] arg (=6)          gzip compression level.
  -x [ --xz-output ]                    compress output data using xz. Adds .xz
                                        extension to output file.
  -u [ --xz-preset ] arg (=6)           xz compression preset level.
  -q [ --query-only ]                   write only query messages to output.
  -r [ --report-info ]                  report info (config and stats summary)
                                        on exit.
  -D [ --relaxed-mode ]                 parse command line allowing
                                        unrecognized options but warning.
  -N [ --no-output ]                    do not output PCAP or template files,
                                        only ancillary files, e.g. info files,
                                        for each input.
  -O [ --no-info ]                      do not output info files.
  -X [ --excludesfile ]                 generate excluded fields files for each
                                        input.
  -S [ --stats ]                        report conversion statistics.
  -k [ --pseudo-anonymisation-key ] arg pseudo-anonymisation key.
  -P [ --pseudo-anonymisation-passphrase ] arg
                                        pseudo-anonymisation passphrase.
  -p [ --pseudo-anonymise ]             pseudo-anonymise output.
  --debug-qr                            print Query/Response details.
```

For example, to generate PCAP output corresponding to the DNS traffic in `capture.cdns`, and automatically `xz` compressing the output:

```
$ inspector -x -o capture.pcap capture.cdns
```

This generates a compressed PCAP output file, `capture.pcap.xz`. It also generates `capture.pcap.info`.

The -q option will only write DNS queries to the output PCAP file.

5.1. *inspector* default_values file

From version 1.0 and later most fields in C-DNS are optional (earlier versions always capture a minimum set of fields). See the *compactor* **excluded_fields** file and the deprecated C-DNS options setting for *compactor* for more details. When converting C-DNS to PCAPs or templated text output it may be necessary or desirable to fill in those omitted fields with default values in a way that was not necessary with earlier versions. Such fields are specified in the **default_values** file.

5.1.1. PCAP generation

The PCAP format requires that certain IP, transport information and basic DNS message contents be present in order for a packet to be considered valid. (Note that only the contents of well-formed DNS messages are captured in the structured C-DNS format used by the DNS-STATS compactor even if not all the data is captured. Whilst the 1.0 C-DNS format supports capture of malformed messages this is not yet implemeted in the DNS-STATS compactor.)

In order for the *inspector* to always be able to generate sane PCAP files from C-DNS with any set of **excluded_fields** a **default_values** file must be present and must contain the full set of required default values. If this file is not present or doesn't contain the required default values *inspector* will exit with an error.

The default values are *only* used if the data field is not present in a particular query/response record, they *never* override data stored in the C-DNS file.

5.1.1.1. Interoperability

A subtlety of the way optionality is specified in the C-DNS format is that even if a field is not 'excluded' in the configuration of the collecting application, it is still not *guaranteed* to be present. For example, a collecting application in another deployment scenario may not have access to the client-hoplimit. Also, in priciple, the presence of a particular field can vary at the individual query/response record level, not just the file level.

Therefore, for maximum interoperability, the **default_values** file is *required* in this release for PCAP regeneration. Its use means that as long as the file is present, a C-DNS file can be processed in its entirety to produce output without failure. Otherwise the absence of a particular field on any single record could cause an error in processing the entire file.

Future releases are likely to include options to allow override of requiring this **default_values** file or for such incomplete records to be ignored in file processing.

5.1.1.2. Corner cases

Whilst the use of **default_values** will produce a 'valid' PCAP file (e.g. one that can be loaded into a processing application such as Wireshark) this does not guarantee that for all permutations

excluded_fields and **default_values** the packets in the file will form a coherent traffic flow with fully reconstructed DNS messages. See the following sections for more details on specific default values and Reconstructed PCAP files for more general issues with PCAP reconstruction.

5.1.2. Template based text output

Since the templated text based output is defined via a template the user should carefully consider which specific defaults to provide to create the required output. For this reason the **default_values** is not a mandatory requirement in this processing flow.

5.1.3. Default_values file location

On startup, *inspector* looks for a **default_values** file. By default this is named `default_values.conf` and is located in a `dns-stats-compactor` system configuration directory. A different location can specified on the command line by using the *inspector* --defaultsfile option.

If installed from a binary package on Linux, a default values file will be at `/usr/local/etc/dns-stats-compactor/default_values.conf`.

5.1.4. Default configuration

The `default_values.conf` file looks like this

```
# A list of default values for fields that are needed but were omitted from the
capture.
# The fields MUST go under the headings as shown

[ip-header]
time-offset=0s                # <n>s|ms|us|ns
response-delay=5ms            # <n>s|ms|us|ns
client-address=127.0.0.1      # IPAddress
client-port=9999              # uint16
client-hoplimit=64            # uint8
server-address=127.0.0.2      # IPAddress
server-port=53                # uint16
server-hoplimit=64            # uint8
qr-transport-flags=ipv4 udp   # (ipv4|ipv6) (udp|tcp) (trailing-data)

[dns-header]
transaction-id=0              # uint16
query-opcode=query            # One of query, iquery, status, notify, update or dso
query-rcode=noerror           # One of any IANA RCODE name, noerror, formerr,
servfail etc.
dns-flags=                    # empty or (query|response)-(cd|ad|z|ra|rd|tc|aa)
query-do
response-rcode=noerror        # One of any IANA RCODE names, noerror, formerr,
servfail etc.

[dns-payload]
query-name=example.com        # As normal text, will be translated to label format
query-class=in                # One of any IANA RR CLASSes, internet, in, chaos, ch,
etc.
query-type=a                  # One of any IANA RR TYPEs, A, NS, CNAME, etc.
rr-ttl=300                    # uint32
query-udp-size=1220           # uint16
query-edns-version=0          # uint8
```

The comments on each line show the format of, and in some cases the set of, allowed values. For example to set a default for dns-flags you could edit the file to say:

```
dns-flags=query-rd response-ra
```

5.1.5. Notes on specific default values

Some details on default values and resulting PCAP files:

- `dns-flags` is the only default that can have no value specified in order to set all flags to 0. It is also the only default that can have multiple values.
- No defaults are provides for excluded RDATA fields since there is no suitable general default (the content is type specific). If the RDATA field was excluded on capture the RDATA field will be

empty in the resulting PCAP.

- If the time-offset was excluded on capture then using the default value (0s) will result in all the query (Q/R=0) traffic having the same timestamp. A similar result occurs for response traffic if the response-delay was excluded on capture.
- Using default timestamps or port numbers can result in traffic flows that are hard to interpret or appear inconsistent, particularly for TCP traffic.
- A corner case exists where so little data was captured in a record that the *inspector* cannot determine if a query and/or response was captured. In this case the *inspector* defaults to generating a query.

A more detailed discussion of the issues around C-DNS to PCAP reconstruction can be found in Section 6.2.1, Section 9 and Appendix D of RFC8618.

5.1.6. Combining excluded_fields, default_values and pcap-filters

Some further subtleties of optionality involve processing of responses. For example, the PCAPs reconstructed by processing C-DNS files captured with the following **exclude_fields**:

```
response-delay
response-rcode
response-answer-sections
response-authority-sections
response-additional-sections
```

will be different to using the following pcap filter:

```
filter=dst host <serverIP>
```

This is because in the first case *compactor* records that it saw a response, even though it doesn't record anything about the response content. It will therefore attempt to reconstruct a response in the PCAP file using the **default_values** (unless the -q option is used). In the latter case *compactor* never saw the response, records that one was not present and does not attempt to reconstruct one.

5.2. Templated text output

By default, *inspector* converts input C-DNS to PCAP. Alternatively, though, it can be used to convert input C-DNS to text output based on a text template file.

5.2.1. Template format

5.2.1.1. Template data items

The text in the template file specifies what should be output for each query/response record in the C-DNS file. Any items in the template enclosed in double braces, known as template markers e.g. `{{NAME}}`, are replaced with a value `NAME` from the query/response (QR) item. Marker values may also be specified on the command line.

A full list of query/response template markers is given in the *inspector* manual page, **inspector**(1). The following table gives a selection of common markers:

`client_address`	IP address of client as raw bytes, 4 for IPv4, 16 for IPv6.
`client_port`	Port used by client.
`server_address`	IP address of server as raw bytes, 4 for IPv4, 16 for IPv6.
`server_port`	Port used by server.
`client_hoplimit`	Value of query client hoplimit. Blank if no query.
`id`	Query ID, or response ID if no query.
`query_name`	QNAME in first Question. Blank if no question.
`query_type`	QTYPE in first Question. Blank if no question.
`query_response_has_query`	1 if QR contains a query, otherwise 0.
`query_response_has_response`	1 if QR contains a response, otherwise 0.
`query_response_query_has_question`	1 if QR contains a query which has a question, otherwise 0. Blank if no query.
`response_rcode`	Value of response RCODE. Blank if no response.
`response_delay_nanosecs`	Nano-seconds between query and response timestamps. 0 if no query or no response.
`timestamp_secs`	QR timestamp seconds since epoch.
`timestamp_microsecs`	QR timestamp micro-seconds since epoch.
`timestamp_nanosecs`	QR timestamp nano-seconds since epoch.
`transport_ipv6`	1 if the IPv4 used, 0 if IPv4.
`transport_tcp`	1 if the transport used was TCP, 0 if UDP.

5.2.1.2. Template marker modifiers

The text substituted into a template marker may be modified by a marker modifier. A marker modifier is a filter that is applied when the template is expanded and modifies the value of the marker before it is output. Modifiers are specified by following the marker name with a colon and the modifier name, e.g. `{{client_address:x-ipaddr}}`.

There is a selection of marker modifiers useful for C-DNS.

`x-cstring`	Output binary input data using C string style escapes, e.g. `Hello, world\n\0`. Non-printable characters without a defined escape are output as `\xaa`.
`x-csvescape`	Output input data escaping for use with CSV as described in RFC4180, e.g. `"Hello, ""quotes"""`.

x-hexstring	Output binary input data as hex characters. A 0 byte is output as \0, all other values as \xaa.
x-ipaddr	Output text representation of a IPv4 or IPv6 address, depending on whether filter input was 4 or 16 bytes.
x-ip6addr	Output IPv4 or IPv6 address as the IPv6 printable address representation. IPv4 addresses are output as ::ffff:192.0.2.25.
x-ip6addr-bin	Output IPv4 or IPv6 addresses as a 16 byte binary IPv6 address. IPv4 addresses are output as 0x00000000000000000000ffffc0000219.
x-date	Output timestamp in seconds as a date in ISO601 format (YYYY-MM-DD).
x-datetime	Output timestamp in seconds as a date and time in ISO601 format (YYYY-MM-DD HH:MM:SS).

5.2.2. Template example

The following template file test.tpl outputs a UTC timestamp, a node identifier (supplied outside C-DNS), the client address and client port and the query name in one query/response as a single record in comma-separated value format (CSV).

```
{{timestamp_secs:x-datetime}},{{node}},{{client_address:x-
ipaddr}},{{client_port}},{{query_name:x-csvescape}}
```

When run with input C-DNS file input.cdns using this command line, it produces the illustrated sample output:

```
$ ./inspector -o - -F template -t ./test.tpl --value node=42 ./input.cdns
2018-02-07 12:38:10,42,173.253.104.205,37529,www.haokan.party.local
2018-02-07 12:38:10,42,19.134.250.211,36396,"36.5.84.123, 10.121.88.32, 10.121.87.219"
2018-02-07 12:38:10,42,242.68.160.159,50139,nllczxstbgfa.local
```

5.3. Pseudo-anonymised output

inspector output may be optionally pseudo-anonymised (see Appendix A for a full description).

Briefly, this means that client and server IP addresses in PCAP or templated text output are pseudo-anonymised, as are all IP addresses in counts of events recorded. IP addresses in data returned from server to client are not anonymised. Client IP address information sent from the client is pseudo-anonymised provided it occurs in locations prescribed in DNS standards.

Pseudo-anonymisation of an IP address takes the original address and passes it through an encoding mechanism to generate a different address. The technical details of the the current implementation are described in Appendix A.

Pseudo-anonymisation is currently an experimental feature. The pseudo-anonymisation services provided and the details of the mechanism used are subject to change.

5.4. Reconstructed PCAP files

The PCAP files generated from C-DNS are not an exact reproduction of the original capture, mostly due to limitations in the C-DNS format itself.

- Link information below the IP layer is not preserved. An Ethernet wrapper is generated for the packets, but MAC addresses are not preserved.
- Queries and responses over TCP will be generated as a TCP stream, but the stream details will not be exactly reproduced.
- Some effort is made to ensure that label compression matches the original, but the details of compression are not recorded in C-DNS and so the match is not perfect. For Knot and NSD nameservers, the error rate is typically in the region of 0.1%. Mismatches are reported as 'REGENERATION ERRORS' in the `info` file.
- IP fragmentation is not preserved.
- If the original C-DNS did not record all DNS message sections, these obviously will not be reproduced.
- Surplus data at the end of a message is not recorded. A count is kept of the number of original packets with surplus data.

5.5. Stored address prefix lengths

compactor C-DNS output may optionally store IP address prefixes rather than the full address when record address events or query/responses. Depending on the capture environment, this may serve to reduce the size of the C-DNS capture.

Separate prefix lengths may be given for client and server addresses. Client addresses are sender addresses on queries and recipient addresses on responses. Server addresses are the reverse; recipient addresses on queries and sender addresses on responses.

The prefix is the number of bits of the address to be stored. Seperate prefixes must be specified for IPv4 addresses and IPv6 addresses. Set prefix lengths using *compactor* configurations `client-address-prefix-ipv4`, `client-address-prefix-ipv6`, `server-address-prefix-ipv4` and `server-address-prefix-ipv6`.

When storing the addresses, *compactor* sets all the address bits after the prefix length to 0, and does not store trailing 0 bytes in the address. So, for example, specifying an IPv6 prefix of 64 will store only the first 8 bytes of the address, rather than the full 16 bytes. Not only does this halve the storage requirement for the address, but it increases the chances of the address being re-used in subsequent records, as only the prefix bits of the address have to match an already-stored address.

5.6. *inspector* limitations

The design of C-DNS allows blocks with different storage and collection parameters to be present in the same C-DNS file. This might arise when two separate C-DNS files are merged.

At the time of writing, no tool exists to do this. *inspector* has not been tested on such inputs, and may not exhibit correct behaviour if presented with such a file. At the time of writing, *inspector* will only display configuration and storage hints from the first set of parameters in the file.

5.7. *compactor/inspector* `info` output

A typical `info` file is as follows. The report begins with information on *compactor* configuration used to capture the data.

```
CONFIGURATION:
  Query timeout         : 5 seconds
  Skew timeout          : 10 microseconds
  Snap length           : 65535
  DNS port              : 53
  Max block items       : 5000
  Promiscuous mode      : Off
  Capture interfaces    :
  Server addresses      :
  VLAN IDs              :
  Filter                :
  Query options         : Extra questions, Answers, Authorities, Additionals
  Response options      : Extra questions, Answers, Authorities, Additionals
  Accept OPCODEs        : QUERY, IQUERY, STATUS, NOTIFY, UPDATE, DSO
  Accept RR types       : A, NS, MD, MF, CNAME, SOA, MB, MG, MR, NULL_R, WKS, PTR,
HINFO, MINFO, MX, TXT, RP, AFSDB, X25, ISDN, RT, NSAP, NSAP_PTR, SIG, KEY, PX, GPOS,
AAAA, LOC, NXT, EID, NIMLOC, SRV, ATMA, NAPTR, KX, CERT, A6, DNAM, SINK, OPT, APL, DS,
SSHFP, IPSECKEY, RRSIG, NSEC, DNSKEY, DHCID, NSEC3, NSEC3PARAM, TLSA, HIP, NINFO,
RKEY, TALINK, CDS, SPF, UINFO, UID, GID, UNSPEC, NID, L32, L64, LP, EU148, EUI64,
TKEY, TSIG, IXFR, AXFR, MAILB, MAILA, TYPE_ANY, URI, CAA, TA
```

There follows information on the program used to created the C-DNS and the host on which it was running.

```
COLLECTOR:
  Collector ID          : dns-stats-compactor 1.1.0
  Collection host ID    : capturehost
```

After that, the `info` file gives time information on the capture. The `Earliest data` and the `Latest data` are the timestamps of the earliest and latest query/response pairs in the file. The reported `Data range` is the time between `Earliest data` and `Latest data`.

```
TIMES:
  Collection started   : 2020-11-02 17h07m49s632204us UTC
  Earliest data        : 2020-11-02 17h07m49s632204us UTC
  Latest data          : 2020-11-02 17h07m59s637124us UTC
  Collection ended     : 2020-11-02 17h08m23s709295us UTC
  Data range           : 10s4920us
  File duration        : 34s77091us
```

When recording traffic from a network interface, *compactor* records the time recording started as the `Collection started` time. When rolling over to a new file during recording, the timestamp of the data that causes the rollover is recorded as the `Collection started` time in the new file. Similarly, when recording is stopped, the time is recorded as the `Collection ended` time. When rolling over to a new file during recording, the timestamp of the data that causes the rollover is recorded as the `Collection ended` time in the old file.

If `Collection started` and `Collection ended` times are available, the time between them is reported as the `File duration`.

If C-DNS is produced directly from a PCAP file, *compactor* cannot know what time the recording started or ended, so no `Collection started` or `Collection ended` times are recorded or reported and therefore no `File duration` is reported.

Recording the `Collection started` and `Collection ended` times is a *compactor* extension to the C-DNS RFC format.

Then follows some overall statistics on the capture.

```
STATISTICS:
  Total Packets received                     : 17493
  Dropped packets at sniffer     (overload) : 0
  Total Packets processed                    : 17493
  Dropped Matcher messages       (overload) : 0
  Discarded C-DNS messages       (sampling) : 0
  Processed DNS messages            (C-DNS) : 16529
  Matched DNS query/response pairs (C-DNS) : 8263
  Unmatched DNS queries             (C-DNS) : 2
  Unmatched DNS responses           (C-DNS) : 1
  Discarded OPCODE DNS messages     (C-DNS) : 0
  Malformed DNS messages            (C-DNS) : 2
  Non-DNS packets                            : 161
  Out-of-order DNS query/responses           : 0
  Dropped raw PCAP packets       (overload) : 0
  Dropped non-DNS packets        (overload) : 0

PCAP STATISTICS:
  Packets received                (libpcap) : 17493
  Packets dropped at i/f          (libpcap) : 0
  Packets dropped in kernel       (libpcap) : 0
```

The above counts are based on traffic as reported to *compactor* by the underlying *libpcap* library. When under load, this may not reflect actual traffic.

When processing from capture files, the `PCAP STATISTICS` will all be 0.

And finally counts of occurrences of various events recorded, and associated addresses.

```
TCP RESETS:
  Count:     1  Address: 3502:e3d3:b836:2ec5:5f1e:98ee:38d8:5cba

ICMP DEST UNREACHABLE:
  Code:  1  Count:     3  Address: 57.98.199.98
  Code:  3  Count:     1  Address: 46.119.7.172
  Code:  3  Count:     1  Address: 158.99.9.124
  Code:  3  Count:     1  Address: 185.158.213.169
  Code:  3  Count:     1  Address: 192.168.59.90
  Code:  3  Count:     1  Address: 214.214.142.170
  Code:  3  Count:     2  Address: 180.243.253.249
  Code:  3  Count:     3  Address: 39.161.250.99
  Code: 13  Count:     4  Address: 215.101.68.68

ICMPv6 DEST UNREACHABLE:
  Code:  1  Count:     4  Address: 3923:96a5:46df:bd71:e9eb:6464:e35:795
  Code:  1  Count:     7  Address: 3923:96a5:46df:bd71:d2ea:2443:527f:55df
  Code:  4  Count:     3  Address: 210a:3c44:990b:d1f0:42a9:bc3f:63e1:240a

====================

REGENERATION ERRORS:
  Incorrect wire size: 11 packets
```

Appendix A: Pseudo-anonymisation

In many jurisdictions IP addresses may, in certain circumstances, be regarded as *personal data* and so data containing IP addresses may be subject to data protection laws.

Pseudo-anonymisation means the processing of personal data in such a way that the data can no longer be attributed to a specific data subject without the use of additional information. However, it is not intended to preclude any other measures of data protection. We do not attempt to provide an exhaustive description of *pseudo-anonymisation* or its limitations here. We recommend that any organisation using this facility fully understand the implications of sharing pseudo-anonymised data for their own use case and independently verify this mechanism meets their needs.

This version of the *inspector* includes an **experimental** facility which applies pseudo-anonymisation to some IP addresses in the PCAP and other outputs from *inspector*. To be exact:

In PCAP output: - Client and server IP addresses in the IP traffic headers. - EDNS(0) Client subnet information in DNS queries from the client.

In `.info` output: - IP addresses of Address Event Counts. - The Server Addresses field

IP addresses supplied by the server as answers to queries from clients are not pseudo-anonymised. Note that only DNS messages are re-generated in PCAP files produced by the *inspector*, no ICMP or other non-DNS messages are generated.

inspector is only able to pseudo-anonymise IP addresses within DNS messages in records that are defined in the DNS standards. Any IP addresses included in non-standard records cannot be reliably distinguished from non-address data, so only addresses in standard locations within records can be processed.

This implementation is experimental and subject to change.

Technical details

IP address pseudo-anonymisation is done by encrypting addresses with AES-128 using a 16 byte key. That key can be supplied directly, via the command line parameter `--pseudo-anonymisation-key`. Alternatively, a key can be generated from a passphrase supplied by the `--pseudo-anonymisation-passphrase` command line parameter.

Many, but not all, aspects of the currently implemented pseudo-anonymisation are similar to the `ipcipher` proposals from PowerDNS. In particular the IPv4 address pseudo-anonymisation is quite different.

Key generation from passphrase

The process for generating a key from a passphrase is to apply PBKDF2 with SHA1 as the hashing function, a salt `cdnscdnscdnscdns`, 50,000 iterations for a 16 byte key. See also RFC2898.

IPv4 address pseudo-anonymisation

IPv4 address pseudo-anonymisation in *inspector* is done using the following process:

1. Fill a 16 byte buffer with 4 concatenated copies of the IPv4 address (4 bytes each).
2. Apply AES-128 to the buffer using the key.
3. Use the most significant 4 bytes of the result (i.e. the first 4 bytes in the buffer) as the pseudo-anonymised IPv4 address.

IPv6 address pseudo-anonymisation

IPv6 address pseudo-anonymisation in *inspector* is done using the following process:

1. Fill a 16 byte buffer with the IPv6 address.
2. Apply AES-128 to the buffer using the key.
3. Use the result as the pseudo-anonymised IPv6 address.

EDNS(0) Client subnet pseudo-anonymisation

EDNS(0) Client subnet addresses described in RFC7871 are pseudo-anonymised using the following process:

1. Depending on the address family indicated in the option, construct an IPv4 or IPv6 address with its significant bits set to the address bits passed in the option and the rest set to 0.
2. Obtain a pseudo-anonymised address based on the constructed address.
3. Set all bits in this address not included in the source prefix length from the option to 0.
4. Replace the option address bits with the significant bits from the pseudo-anonymised address.