POWERDNS

dnsdist

PowerDNS.COM BV

Apr 25, 2024

CONTENTS

CHAPTER

ONE

DNSDIST OVERVIEW

dnsdist is a highly DNS-, DoS- and abuse-aware loadbalancer. Its goal in life is to route traffic to the best server, delivering top performance to legitimate users while shunting or blocking abusive traffic.

dnsdist is dynamic, its configuration language is Lua and it can be changed at runtime, and its statistics can be queried from a console-like interface or an HTTP API.

A configuration to balance DNS queries to several backend servers:

```
newServer({address="2620:fe::fe", qps=1})
newServer({address="2620:fe::9", qps=1})
newServer({address="9.9.9.9", qps=1})
newServer({address="2001:db8::1", qps=10})
newServer({address="[2001:db8::2]:5300", name="dns1", qps=10})
newServer("192.0.2.1")
setServerPolicy(firstAvailable) -- first server within its QPS limit
```

1.1 Running dnsdist

If you have not worked with dnsdist before, here are some resources to get you going:

- *Install dnsdist*.
- To get a feeling for how it works, see the *Quickstart Guide*.
- *Running and Configuring dnsdist*
- The *Packet Policies* page covers how to apply policies to traffic
- There are several *Guides* about the different features and options
- *Advanced Topics* describes some of the more advanced features
- *Reference Guides* has all the configuration and object information

1.2 Questions, requests or comments?

There are several ways to reach us:

- The dnsdist mailing-list
- #powerdns on irc.oftc.net

The Open-Xchange/PowerDNS company can provide help or support you in private as well. Please contact Open-Xchange.

This documentation is also available as a PDF document.

CHAPTER

TWO

INSTALLING DNSDIST

dnsdist only runs on UNIX-like systems and there are several ways to install dnsdist. The fastest way is using packages, either from your own operating system vendor or supplied by the PowerDNS project. Building from source is also supported.

2.1 Installing from Packages

If dnsdist is available in your operating system's software repositories, install it from there. However, the version of dnsdist in the repositories might be an older version that might not have a feature that was added in a later version. Or you might want to be brave and try a development snapshot from the master branch. PowerDNS provides software repositories for the most popular distributions. Visit https://repo.powerdns.com for more information and installation instructions.

2.1.1 Debian

For Debian and its derivatives (like Ubuntu) installing the `dnsdist` package should do it:

```
apt-get install -y dnsdist
```

2.1.2 Red Hat

For Red Hat, CentOS and its derivatives, dnsdist is available in EPEL:

```
yum install -y epel-release
yum install -y dnsdist
```

2.1.3 FreeBSD

dnsdist is also available in FreeBSD ports.

2.2 Installing from Source

In order to compile dnsdist, a modern compiler with C++ 2017 support and GNU make are required. dnsdist depends on the following libraries:

- Boost
- Lua 5.1+ or LuaJit
- Editline (libedit)
- libfstrm (optional, dnstap support)

- GnuTLS (optional, DoT and outgoing DoH support)
- libbpf and libxdp (optional, *XSK/AF_XDP* support)
- libcap (optional, capabilities support)
- libh2o (optional, incoming DoH support, deprecated in 1.9.0 in favor of `nghttp2`)
- libsodium (optional, DNSCrypt and console encryption support)
- LMDB (optional, LMDB support)
- net-snmp (optional, SNMP support)
- nghttp2 (optional, outgoing DoH support)
- OpenSSL (optional, DoT and DoH support)
- protobuf (optional, not needed as of 1.6.0)
- quiche (optional, incoming DoQ support)
- re2 (optional)
- TinyCDB (optional, CDB support)

Should **dnsdist** be run on a system with systemd, it is highly recommended to have the systemd header files (`libsystemd-dev` on Debian and `systemd-devel` on CentOS) installed to have **dnsdist** support `systemd-notify`.

2.2.1 From tarball

Release tarballs are available from the downloads site, snapshot and pre-release tarballs can be found as well.

The release tarballs have detached PGP signatures, signed by one of these PGP keys:

- FBAE 0323 821C 7706 A5CA 151B DCF5 13FA 7EED 19F3
- D630 0CAB CBF4 69BB E392 E503 A208 ED4F 8AF5 8446
- 16E1 2866 B773 8C73 976A 5743 6FFC 3343 9B0D 04DF
- 990C 3D0E AC7C 275D C6B1 8436 EACA B90B 1963 EC2B

There is a PGP keyblock with these keys available on https://dnsdist.org/_static/dnsdist-keyblock.asc.

Older (1.0.x) releases can also be signed with one of the following keys:

- 1628 90D0 689D D12D D33E 4696 1C5E E990 D2E7 1575
- B76C D467 1C09 68BA A87D E61C 5E50 715B F2FF E1A7
- Untar the tarball and `cd` into the source directory
- Run `./configure`
- Run `make` or `gmake` (on BSD)

2.2.2 From git

To compile from git, these additional dependencies are required:

- GNU Autoconf
- GNU Automake
- Ragel

dnsdist source code lives in the PowerDNS git repository but is independent of PowerDNS.

```
git clone https://github.com/PowerDNS/pdns.git
cd pdns/pdns/dnsdistdist
autoreconf -i
./configure
make
```

2.2.3 OS Specific Instructions

None, really.

2.2.4 Build options

Our `configure` script provides a fair number of options with regard to which features should be enabled, as well as which libraries should be used. In addition to these options, more features can be disabled at compile-time by defining the following symbols:

- `DISABLE_BUILTIN_HTML` removes the built-in web pages
- `DISABLE_CARBON` for carbon support
- `DISABLE_COMPLETION` for completion support in the console
- `DISABLE_DELAY_PIPE` removes the ability to delay UDP responses
- `DISABLE_DEPRECATED_DYNBLOCK` for legacy dynamic blocks not using the new `DynBlockRulesGroup` interface
- `DISABLE_DYNBLOCKS` disables the new dynamic block interface
- `DISABLE_ECS_ACTIONS` to disable actions altering EDNS Client Subnet
- `DISABLE_FALSE_SHARING_PADDING` to disable the padding of atomic counters, which is inserted to prevent false sharing but increases the memory use significantly
- `DISABLE_HASHED_CREDENTIALS` to disable password-hashing support
- `DISABLE_LUA_WEB_HANDLERS` for custom Lua web handlers support
- `DISABLE_OCSP_STAPLING` for OCSP stapling
- `DISABLE_OPENSSL_ERROR_STRINGS` to disable the loading of OpenSSL's error strings, reducing the memory use at the cost of human-readable error messages
- `DISABLE_NPN` for Next Protocol Negotiation, superseded by ALPN
- `DISABLE_PROMETHEUS` for prometheus
- `DISABLE_PROTOBUF` for protocol-buffer support, including dnstap
- `DISABLE_RECVMMSG` for `recvmmsg` support
- `DISABLE_RULES_ALTERING_QUERIES` to remove rules altering the content of queries
- `DISABLE_SECPOLL` for security polling
- `DISABLE_WEB_CACHE_MANAGEMENT` to disable cache management via the API
- `DISABLE_WEB_CONFIG` to disable accessing the configuration via the web interface

Additionally several Lua bindings can be removed when they are not needed, as they increase the memory required during compilation and the size of the final binary:

- `DISABLE_CLIENT_STATE_BINDINGS`
- `DISABLE_COMBO_ADDR_BINDINGS`
- `DISABLE_DNSHEADER_BINDINGS`

- DISABLE_DNSNAME_BINDINGS
- DISABLE_DOWNSTREAM_BINDINGS
- DISABLE_NETMASK_BINDINGS
- DISABLE_NON_FFI_DQ_BINDINGS
- DISABLE_PACKETCACHE_BINDINGS
- DISABLE_POLICIES_BINDINGS
- DISABLE_QPS_LIMITER_BINDINGS
- DISABLE_SUFFIX_MATCH_BINDINGS
- DISABLE_TOP_N_BINDINGS

Finally a build flag can be used to make use a single thread to handle all incoming UDP queries from clients, no matter how many *addLocal()* directives are present in the configuration. It also moves the task of accepting incoming TCP connections to the TCP workers themselves, removing the TCP acceptor threads. This option is destined to resource-constrained environments where dnsdist needs to listen on several addresses, over several interfaces, and one thread is enough to handle the traffic and therefore the overhead of using multiples threads for that task does not make sense. This option can be enabled by setting USE_SINGLE_ACCEPTOR_THREAD.

CHAPTER

THREE

QUICKSTART GUIDE

This guide gives an overview of dnsdist features and operations.

3.1 Running in the Foreground

After *installing* dnsdist, the quickest way to start experimenting is launching it on the foreground with:

```
dnsdist -l 127.0.0.1:5300 9.9.9.9 2620:fe::fe 2620:fe::9
```

This will make dnsdist listen on IP address 127.0.0.1, port 5300 and forward all queries to the three listed IP addresses, with a sensible balancing policy.

3.2 dnsdist Console and Configuration

Here is more complete configuration, save it to dnsdist.conf:

```
newServer({address="2001:db8::1", qps=1})
newServer({address="2001:db8::2", qps=1})
newServer({address="[2001:db8::3]:5300", qps=10})
newServer({address="2001:db8::4", name="dns1", qps=10})
newServer("192.0.2.1")
setServerPolicy(firstAvailable) -- first server within its QPS limit
```

The *newServer()* function is used to add a backend server to the configuration.

Now run dnsdist again, reading this configuration:

```
$ dnsdist -C dnsdist.conf --local=0.0.0.0:5300
Marking downstream [2001:db8::1]:53 as 'up'
Marking downstream [2001:db8::2]:53 as 'up'
Marking downstream [2001:db8::3]:5300 as 'up'
Marking downstream [2001:db8::4]:53 as 'up'
Marking downstream 192.0.2.1.:53 as 'up'
Listening on 0.0.0.0:5300
>
```

You can now send queries to port 5300, and get answers:

```
$ dig -t aaaa powerdns.com @127.0.0.1 -p 5300 +short +nocookie
2001:888:2000:1d::2
```

Note that dnsdist dropped us in a prompt above, where we can get some statistics:

```
> showServers()
#   Address                    State      Qps     Qlim Ord Wt     Queries   Drops␣
↪Drate    Lat Pools
0   [2001:db8::1]:53              up      0.0        1   1  1           1       0   0.
↪0   0.0
1   [2001:db8::2]:53              up      0.0        1   1  1           0       0   0.
↪0   0.0
2   [2001:db8::3]:5300            up      0.0       10   1  1           0       0   0.
↪0   0.0
3   [2001:db8::4]:53              up      0.0       10   1  1           0       0   0.
↪0   0.0
4   192.0.2.1:53                  up      0.0        0   1  1           0       0   0.
↪0   0.0
All                                       0.0                           1       0
```

showServers() is usually one of the first commands you will use when logging into the console. More advanced topics are covered in *Working with the dnsdist Console*.

Here we also see our configuration. 5 downstream servers have been configured, of which the first 4 have a QPS limit (of 1, 1, 10 and 10 queries per second, respectively).

The final server has no limit, which we can easily test:

```
$ for a in {0..1000}; do dig powerdns.com @127.0.0.1 -p 5300 +noall +nocookie > /
↪dev/null; done
```

```
> showServers()
#   Address                    State      Qps     Qlim Ord Wt     Queries   Drops␣
↪Drate    Lat Pools
0   [2001:db8::1]:53              up      1.0        1   1  1           7       0   0.
↪0   1.6
1   [2001:db8::2]:53              up      1.0        1   1  1           6       0   0.
↪0   0.6
2   [2001:db8::3]:5300            up     10.3       10   1  1          64       0   0.
↪0   2.4
3   [2001:db8::4]:53              up     10.3       10   1  1          63       0   0.
↪0   2.4
4   192.0.2.1:53                  up    125.8        0   1  1         671       0   0.
↪0   0.4
All                                     145.0                         811       0
```

Note that the first 4 servers were all limited to near their configured QPS, and that our final server was taking up most of the traffic. No queries were dropped, and all servers remain up.

3.2.1 Changing Server Settings

The servers from *showServers()* are numbered, *getServer()* is used to get this *Server* object to manipulate it.

To force a server down, try *Server:setDown()*:

```
> getServer(0):setDown()
> showServers()
#   Address                    State      Qps     Qlim Ord Wt     Queries   Drops␣
↪Drate    Lat Pools
0   [2001:db8::1]:53            DOWN      0.0        1   1  1           8       0   0.
↪0   0.0
...
```

The DOWN in all caps means it was forced down. A lower case down would've meant that dnsdist itself had concluded the server was down. Similarly, *Server:setUp()* forces a server to be up, and *Server:setAuto()* returns it to the default availability-probing.

To change the QPS for a server, use *Server:setQPS()*:

```
> getServer(0):setQPS(1000)
```

3.3 Restricting Access

By default, dnsdist listens on `127.0.0.1` (not `::1`!), port 53.

To listen on a different address, use the `-l` command line option (useful for testing in the foreground), or use *setLocal()* and *addLocal()* in the configuration file:

```
setLocal('192.0.2.53')      -- Listen on 192.0.2.53, port 53
addLocal('[::1]:5300') -- Also listen on ::1, port 5300
```

Before packets are processed they have to pass the ACL, which helpfully defaults to **RFC 1918** private IP space. This prevents us from easily becoming an open DNS resolver.

Adding network ranges to the *ACL* is done with the *setACL()* and *addACL()* functions:

```
setACL({'192.0.2.0/28', '2001:db8:1::/56'}) -- Set the ACL to only allow these␣
↪subnets
addACL('2001:db8:2::/56')                   -- Add this subnet to the existing ACL
```

3.4 Securing the path to the backend

dnsdist has always been designed as a load-balancer placed in front of authoritative or recursive servers, assuming that the network path between dnsdist and these servers is trusted.

If dnsdist is instead intended to be deployed in such a way that the path to its backend is not secure, the UDP protocol should not be used, and 'TCP-only', DNS over TLS and DNS over HTTPS protocols used instead, as supported since 1.7.0.

For more details, please look at the *Configuring Downstream Servers* guide.

3.5 More Information

Following this quickstart guide allowed you to set up a basic balancing dnsdist instance. However, dnsdist is much more powerful. See the *Guides* and/or the *Advanced Topics* sections on how to shape, shut and otherwise manipulate DNS traffic.

CHAPTER

FOUR

RUNNING AND CONFIGURING DNSDIST

dnsdist is meant to run as a daemon. As such, distribution native packages know how to stop/start themselves using operating system services.

It is configured with a configuration file called `dnsdist.conf` The default path to this file is determined by the `SYSCONFDIR` variable during compilation. Most likely this path is `/etc/dnsdist`, `/etc` or `/usr/local/etc/`, dnsdist will tell you on startup which file it reads.

dnsdist is designed to (re)start almost instantly. But to prevent downtime when changing configuration, the console (see *Working with the dnsdist Console*) can be used for live configuration.

Issuing `delta()` on the console will print the changes to the configuration that have been made since startup:

```
> delta()
-- Wed Feb 22 2017 11:31:44 CET
addLocal('127.0.0.1:5301', false)
-- Wed Feb 22 2017 12:03:48 CET
addACL('192.0.2.1/8')
-- Wed Feb 22 2017 12:05:51 CET
addACL('2001:db8::1')
```

These commands can be copied to the configuration file, should they need to persist after a restart.

4.1 Running as unprivileged user

`dnsdist` can drop privileges using the `--uid` and `--gid` command line switches to ensure it does not run with root privileges. Note that **`dnsdist`** drops its privileges **after** parsing its startup configuration and binding its listening and initial `newServer()` sockets as user *root*. It is highly recommended to create a system user and group for **`dnsdist`**. Note that most packaged versions of **`dnsdist`** already create this user.

4.2 Understanding how queries are forwarded to backends

Initially dnsdist tried to forward a query to the backend using the same protocol than the client used to contact dnsdist: queries received over UDP were forwarded over UDP, and the same for TCP. When incoming DNSCrypt and DNS over TLS support were added, the same logic was applied, so DoT queries are forwarded over TCP. For DNS over HTTPS, UDP was selected instead for performance reason, breaking with the existing logic.

Before 1.7.0, which introduced TCP fallback, that meant that there was a potential issue with very large answers and DNS over HTTPS, requiring careful configuration of the path between dnsdist and the backend. More information about that is available in the *DNS over HTTPS section*.

In addition to TCP fallback for DoH, 1.7.0 introduced three new notions:

- TCP-only backends, for which queries will always forwarded over a TCP connection (see the *tcpOnly* parameter of `newServer()`)

- DNS over HTTPS backends, for which queries are forwarded over a DNS over HTTPS connection (see the *dohPath* parameter of `newServer()`)
- and DNS over TLS backends, for which queries are forwarded over a DNS over TLS connection (see the *tls* parameter of `newServer()`)

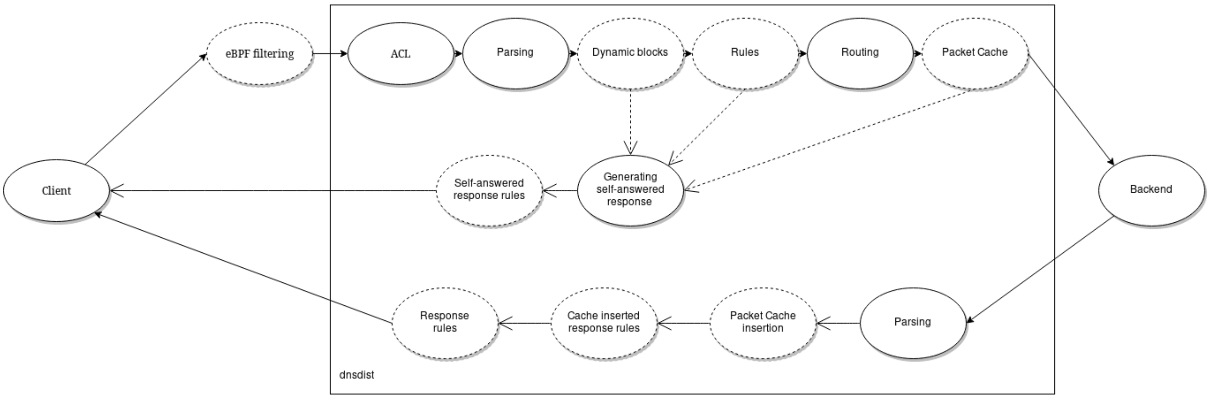
To sum it up:

Incoming	Outgoing (regular)	Outgoing (TCP-only, 1.7+)	Outgoing (TLS, 1.7+)	Outgoing (DoH, 1.7+)
UDP	UDP	TCP	TLS	DoH
TCP	TCP	TCP	TLS	DoH
DNSCrypt UDP	UDP	TCP	TLS	DoH
DNSCrypt TCP	TCP	TCP	TLS	DoH
DoT	TCP	TCP	TLS	DoH
DoH	**UDP**	TCP	TLS	DoH
DoQ	TCP	TCP	TLS	DoH
DoH3	TCP	TCP	TLS	DoH

CHAPTER

FIVE

PACKET POLICIES



dnsdist works in essence like any other loadbalancer:

It receives packets on one or several addresses it listens on, and determines whether it will process this packet based on the *Access Control*. Should the packet be processed, **dnsdist** attempts to match any of the configured rules in order and when one matches, the associated action is performed.

These rule and action combinations are considered policies. The complete list of selectors (rules) can be found in *Rule selectors*, and the list of actions in *Rule Actions*.

5.1 Packet Actions

Each packet can be:

- Dropped
- Turned into an answer directly
- Forwarded to a downstream server
- Modified and forwarded to a downstream and be modified back
- Be delayed

This decision can be taken at different times during the forwarding process. All packets not handled by an explicit action are forwarded to a downstream server in the default pool.

5.1.1 Examples

Rules for traffic exceeding QPS limits

Traffic that exceeds a QPS limit, in total or per IP (subnet) can be matched by a rule.

For example:

```
addAction(MaxQPSIPRule(5, 32, 48), DelayAction(100))
```

This measures traffic per IPv4 address and per /48 of IPv6, and if traffic for such an address (range) exceeds 5 qps, it gets delayed by 100ms. (Please note: `DelayAction()` can only delay UDP traffic).

As another example:

```
addAction(MaxQPSIPRule(5), SetNoRecurseAction())
```

This strips the Recursion Desired (RD) bit from any traffic per IPv4 or IPv6 /64 that exceeds 5 qps. This means any those traffic bins is allowed to make a recursor do 'work' for only 5 qps.

If this is not enough, try:

```
addAction(MaxQPSIPRule(5), DropAction())
```

or:

```
addAction(AndRule{MaxQPSIPRule(5), TCPRule(false)}, TCAction())
```

This will respectively drop traffic exceeding that 5 QPS limit per IP or range, or return it with TC=1, forcing clients to fall back to TCP.

In that last one, note the use of `TCPRule()`. Without it, clients would get TC=1 even if they correctly fell back to TCP.

To turn this per IP or range limit into a global limit, use `NotRule(MaxQPSRule(5000))` instead of `MaxQPSIPRule()`.

Regular Expressions

`RegexRule()` matches a regular expression on the query name, and it works like this:

```
addAction(RegexRule("[0-9]{5,}"), DelayAction(750)) -- milliseconds
addAction(RegexRule("[0-9]{4,}\\.example$"), DropAction())
```

This delays any query for a domain name with 5 or more consecutive digits in it. The second rule drops anything with more than 4 consecutive digits within a .example domain.

Note that the query name is presented without a trailing dot to the regex. The regex is applied case insensitively.

Alternatively, if compiled in, `RE2Rule()` provides similar functionality, but against libre2.

Note that to check if a name is in a list of domains, `QNameSuffixRule()` is preferred over complex regular expressions or multiple instances of `RegexRule()`.

5.2 Managing Rules

Active Rules can be shown with `showRules()` and removed with `rmRule()`:

```
> addAction("h4xorbooter.xyz.", QPSAction(10))
> addAction({"130.161.0.0/16", "145.14.0.0/16"} , QPSAction(20))
> addAction({"nl.", "be."}, QPSAction(1))
> showRules()
#     Matches Rule                                               Action
0           0 h4xorbooter.xyz.                                   qps limit to 10
1           0 130.161.0.0/16, 145.14.0.0/16                      qps limit to 20
2           0 nl., be.                                           qps limit to 1
```

See *Rules management* for more information.

CHAPTER

SIX

STATISTICS

`dnsdist` keeps statistics on the queries it receives and send out. They can be accessed in different ways:

- via the console (see *Working with the dnsdist Console*), using `dumpStats()` for the general ones, `showServers()` for the ones related to the backends, `showBinds()` for the frontends, *getPool("pool name"):getCache():printStats()* for the ones related to a specific cache and so on
- via the internal webserver (see *Built-in webserver*) which includes a Prometheus endpoint
- via Carbon / Graphite / Metronome export (see *Exporting statistics via Carbon*)
- via SNMP (see *SNMP support*)

To make sense of the statistics, the following relation should hold:

> queries - noncompliant-queries = responses - noncompliant-responses + downstream-timeouts + no-policy + rule-drop

Before 1.8.0, cache hits and self-answered responses were not accounted in the responses counters, so the relation was:

> responses - noncompliant-responses + cache-hits + downstream-timeouts + self-answered + no-policy + rule-drop

Note that packets dropped by eBPF (see *eBPF Socket Filtering*) are accounted for in the eBPF statistics, and do not show up in the metrics described on this page.

Note that counters that come from `/proc/net/` are operating system specific counters. They do not reset on service restart and they are not only related to **`dnsdist`**. For more information on these counters, refer to Linux networking counter documentation and the RFC1213.

6.1 acl-drops

The number of packets (or TCP messages) dropped because of the *ACL*. If a packet or message is dropped, it is not counted in the *queries* statistic.

6.2 cache-hits

Number of times a response was sent using data found in the *packet cache*.

6.3 cache-misses

Number of times an answer was not found in the *packet cache*. Only counted if a packet cache was setup for the selected pool.

6.4 cpu-iowait

New in version 1.5.0.

Time spent waiting for I/O to complete by the whole system, in units of USER_HZ.

6.5 cpu-steal

New in version 1.5.0.

Stolen time, which is the time spent by the whole system in other operating systems when running in a virtualized environment, in units of USER_HZ.

6.6 cpu-sys-msec

Milliseconds spent by **`dnsdist`** in the “system” state.

6.7 cpu-user-msec

Milliseconds spent by **`dnsdist`** in the “user” state.

6.8 doh-query-pipe-full

Number of queries dropped because the internal DoH pipe was full.

6.9 doh-response-pipe-full

Number of responses dropped because the internal DoH pipe was full.

6.10 doq-response-pipe-full

Number of responses dropped because the internal DoQ pipe was full.

6.11 downstream-send-errors

Number of errors when sending a query to a backend.

6.12 downstream-timeouts

Number of queries not answer in time by a backend.

6.13 dyn-block-nmg-size

Number of dynamic blocks entries.

6.14 dyn-blocked

Number of queries dropped because of a dynamic block.

6.15 empty-queries

Number of empty queries received from clients. Every empty-query is also counted as a *query*.

6.16 fd-usage

Number of currently used file descriptors.

6.17 frontend-noerror

Number of NoError answers sent to clients.

6.18 frontend-nxdomain

Number of NXDomain answers sent to clients.

6.19 frontend-servfail

Number of ServFail answers sent to clients.

6.20 latency-avg100

Average response latency in microseconds of the last 100 packets received over UDP.

6.21 latency-avg1000

Average response latency in microseconds of the last 1000 packets received over UDP.

6.22 latency-avg10000

Average response latency in microseconds of the last 10000 packets received over UDP.

6.23 latency-avg1000000

Average response latency in microseconds of the last 1000000 packets received over UDP.

6.24 latency-bucket

Histogram of response time latencies for queries received over UDP.

6.25 latency-count

Number of queries contributing to response time histogram and latency sum.

6.26 latency-doh-avg100

Average response latency, in microseconds, of the last 100 packets received over DoH.

6.27 latency-doh-avg1000

Average response latency, in microseconds, of the last 1000 packets received over DoH.

6.28 latency-doh-avg10000

Average response latency, in microseconds, of the last 10000 packets received over DoH.

6.29 latency-doh-avg1000000

Average response latency, in microseconds, of the last 1000000 packets received over DoH.

6.30 latency-doq-avg100

Average response latency, in microseconds, of the last 100 packets received over DoQ.

6.31 latency-doq-avg1000

Average response latency, in microseconds, of the last 1000 packets received over DoQ.

6.32 latency-doq-avg10000

Average response latency, in microseconds, of the last 10000 packets received over DoQ.

6.33 latency-doq-avg1000000

Average response latency, in microseconds, of the last 1000000 packets received over DoQ.

6.34 latency-dot-avg100

Average response latency, in microseconds, of the last 100 packets received over DoT.

6.35 latency-dot-avg1000

Average response latency, in microseconds, of the last 1000 packets received over DoT.

6.36 latency-dot-avg10000

Average response latency, in microseconds, of the last 10000 packets received over DoT.

6.37 latency-dot-avg1000000

Average response latency, in microseconds, of the last 1000000 packets received over DoT.

6.38 latency-slow

Number of queries received over UDP answered in more than 1 second.

6.39 latency-sum

Total response time of all queries received over UDP combined in milliseconds since the start of **dnsdist**. Can be used to calculate the average response time over all queries received over UDP.

6.40 latency-tcp-avg100

Average response latency, in microseconds, of the last 100 packets received over TCP.

6.41 latency-tcp-avg1000

Average response latency, in microseconds, of the last 1000 packets received over TCP.

6.42 latency-tcp-avg10000

Average response latency, in microseconds, of the last 10000 packets received over TCP.

6.43 latency-tcp-avg1000000

Average response latency, in microseconds, of the last 1000000 packets received over TCP.

6.44 latency0-1

Number of queries received over UDP answered in less than 1 ms.

6.45 latency1-10

Number of queries received over UDP answered in 1-10 ms.

6.46 latency10-50

Number of queries received over UDP answered in 10-50 ms.

6.47 latency50-100

Number of queries received over UDP answered in 50-100 ms.

6.48 latency100-1000

Number of queries received over UDP answered in 100-1000 ms.

6.49 no-policy

Number of queries dropped because no server was available.

6.50 noncompliant-queries

Number of queries dropped as non-compliant.

6.51 noncompliant-responses

Number of answers from a backend dropped as non-compliant.

6.52 outgoing-doh-query-pipe-full

Number of outgoing DoH queries dropped because the internal pipe used to distribute queries was full.

6.53 proxy-protocol-invalid

New in version 1.6.0.

Number of queries dropped because of an invalid Proxy Protocol header.

6.54 queries

Number of received queries.

6.55 rdqueries

Number of received queries with the recursion desired bit set.

6.56 real-memory-usage

Current memory usage.

6.57 responses

Number of response sent to clients.

Before 1.8.0, it was the number of responses received from backends, not accounting for cache hits or self-answered responses.

6.58 rule-drop

Number of queries dropped because of a rule.

6.59 rule-nxdomain

Number of NXDomain answers returned because of a rule.

6.60 rule-refused

Number of Refused answers returned because of a rule.

6.61 rule-servfail

Number of ServFail answers returned because of a rule.

6.62 rule-truncated

New in version 1.6.0.

Number of truncated answers returned because of a rule.

6.63 security-status

The security status of **dnsdist**. This is regularly polled.

- 0 = Unknown status or unreleased version
- 1 = OK
- 2 = Upgrade recommended
- 3 = Upgrade required (most likely because there is a known security issue)

6.64 self-answered

Number of self-answered responses.

6.65 servfail-responses

Number of servfail answers received from backends.

6.66 tcp-cross-protocol-query-pipe-full

Number of TCP cross-protocol queries dropped because the internal pipe used to distribute queries was full.

6.67 tcp-cross-protocol-response-pipe-full

Number of TCP cross-protocol responses dropped because the internal pipe used to distribute queries was full.

6.68 tcp-listen-overflows

New in version 1.6.0.

From `/proc/net/netstat` `ListenOverflows`.

6.69 tcp-query-pipe-full

Number of TCP queries dropped because the internal pipe used to distribute queries was full.

6.70 trunc-failures

Number of errors encountered while truncating an answer.

6.71 udp-in-csum-errors

New in version 1.7.0.

From `/proc/net/snmp` `InErrors`.

6.72 udp-in-errors

New in version 1.5.0.

From `/proc/net/snmp` `InErrors`.

6.73 udp-noport-errors

New in version 1.5.0.

From `/proc/net/snmp` `NoPorts`.

6.74 udp-recvbuf-errors

New in version 1.5.0.

From `/proc/net/snmp` `RcvbufErrors`.

6.75 udp-sndbuf-errors

New in version 1.5.0.

From `/proc/net/snmp` `SndbufErrors`.

6.76 udp6-in-csum-errors

New in version 1.7.0.

From `/proc/net/snmp6` `InErrors`.

6.77 udp6-in-errors

New in version 1.7.0.

From `/proc/net/snmp6` `InErrors`.

6.78 udp6-noport-errors

New in version 1.7.0.

From `/proc/net/snmp6` `NoPorts`.

6.79 udp6-recvbuf-errors

New in version 1.7.0.

From `/proc/net/snmp6` `RcvbufErrors`.

6.80 udp6-sndbuf-errors

New in version 1.7.0.

From `/proc/net/snmp6` `SndbufErrors`.

6.81 uptime

Uptime of the **`dnsdist`** process, in seconds.

CHAPTER

SEVEN

CACHING RESPONSES

dnsdist implements a simple but effective packet cache, not enabled by default. It is enabled per-pool, but the same cache can be shared between several pools. The first step is to define a cache with *newPacketCache()*, then to assign that cache to the chosen pool, the default one being represented by the empty string:

```
pc = newPacketCache(10000, {maxTTL=86400, minTTL=0, temporaryFailureTTL=60,␣
↪staleTTL=60, dontAge=false})
getPool(""):setCache(pc)
```

- The first parameter (10000) is the maximum number of entries stored in the cache, and is the only one required. All the other parameters are optional and in seconds, except the last one which is a boolean.
- The second one (86400) is the maximum lifetime of an entry in the cache.
- The third one (0) is the minimum TTL an entry should have to be considered for insertion in the cache.
- The fourth one (60) is the TTL used for a Server Failure or a Refused response.
- The fifth one (60) is the TTL that will be used when a stale cache entry is returned.
- The sixth one is a boolean that when set to true, avoids reducing the TTL of cached entries.

For performance reasons the cache will pre-allocate buckets based on the maximum number of entries, so be careful to set the first parameter to a reasonable value. Something along the lines of a dozen bytes per pre-allocated entry can be expected on 64-bit. That does not mean that the memory is completely allocated up-front, the final memory usage depending mostly on the size of cached responses and therefore varying during the cache's lifetime. Assuming an average response size of 512 bytes, a cache size of 10000000 entries on a 64-bit host with 8GB of dedicated RAM would be a safe choice.

The *setStaleCacheEntriesTTL()* directive can be used to allow dnsdist to use expired entries from the cache when no backend is available. Only entries that have expired for less than n seconds will be used, and the returned TTL can be set when creating a new cache with *newPacketCache()*.

A reference to the cache affected to a specific pool can be retrieved with:

```
getPool("poolname"):getCache()
```

And removed with:

```
getPool("poolname"):unsetCache()
```

Cache usage stats (hits, misses, deferred inserts and lookups, collisions) can be displayed by using the *PacketCache:printStats()* method:

```
getPool("poolname"):getCache():printStats()
```

The same values can also be returned as a Lua table, which is easier to work with from a script, using the *PacketCache:getStats()* method.

Expired cached entries can be removed from a cache using the *PacketCache:purgeExpired()* method, which will remove expired entries from the cache until at most n entries remain in the cache. For example, to remove all expired entries:

```
getPool("poolname"):getCache():purgeExpired(0)
```

Specific entries can also be removed using the *PacketCache:expungeByName()* method:

```
getPool("poolname"):getCache():expungeByName(newDNSName("powerdns.com"), DNSQType.
↪A)
```

Changed in version 1.4.0: Before 1.4.0, the QTypes were in the `dnsdist` namespace. Use `dnsdist.A` in these versions.

Finally, the *PacketCache:expunge()* method will remove all entries until at most n entries remain in the cache:

```
getPool("poolname"):getCache():expunge(0)
```

CHAPTER

EIGHT

EXPORTING STATISTICS VIA CARBON

8.1 Setting up a carbon export

To emit metrics to Graphite, or any other software supporting the Carbon protocol, use:

```
carbonServer('ip-address-of-carbon-server', 'ourname', 30, 'dnsdist', 'main')
```

Where `ourname` can be used to override your hostname, and `30` is the reporting interval in seconds. `dnsdist` and `main` are used as namespace and instance variables. For querycount statistics these two variables are currently ignored. The last four arguments can be omitted. The latest version of PowerDNS Metronome comes with attractive graphs for dnsdist by default.

8.2 Query counters

In addition to other metrics, it is possible to send per-records statistics of the amount of queries by using `setQueryCount()`. With query counting enabled, dnsdist will increase a counter for every unique record or the behaviour you define in a custom Lua function by setting `setQueryCountFilter()`. This filter can decide whether to keep count on a query at all or rewrite for which query the counter will be increased. An example of a QueryCountFilter would be:

```
function filter(dq)
  qname = dq.qname:toString()

  -- don't count PTRs at all
  if(qname:match('in%-addr.arpa$')) then
    return false, ""
  end

  -- count these queries as if they were queried without leading www.
  if(qname:match('^www.')) then
    qname = qname:gsub('^www.', '')
  end

  -- count queries by default
  return true, qname
end

setQueryCountFilter(filter)
```

Valid return values for `QueryCountFilter` functions are:

- true: count the specified query
- false: don't count the query

Note that the query counters are buffered and flushed each time statistics are sent to the carbon server. The current content of the buffer can be inspected with :`getQueryCounters()`. If you decide to enable query counting without *carbonServer()*, make sure you implement clearing the log from `maintenance()` by issuing `clearQueryCounters()`.

CHAPTER

NINE

WORKING WITH THE DNSDIST CONSOLE

dnsdist can expose a commandline console over an encrypted tcp connection for controlling it, debugging DNS issues and retrieving statistics.

The console can be enabled with *controlSocket()*:

```
controlSocket('192.0.2.53:5199')
```

Enabling the console without encryption enabled is not recommended. Note that encryption requires building dnsdist with either libsodium or libcrypto support enabled.

Once you have a console-enabled dnsdist, the first step to enable encryption is to generate a key with *makeKey()*:

```
$ ./dnsdist -l 127.0.0.1:5300 -C /dev/null
[..]
> makeKey()
setKey("ENCODED KEY")
```

The example above tells **dnsdist** not to load the default configuration file (`-C /dev/null`) to prevent it from trying to listen on privileged ports, connect to backends, etc. It also instructs **dnsdist** not to listen on the default (privileged) port 53 of all available addresses but on an unprivileged and hopefully available port 5300 on the local interface instead (`-l 127.0.0.1:5300`).

The key does not have a specific format, so base-64 encoding 32 random bytes works as well:

```
$ dd if=/dev/random bs=1 count=32 status=none | base64
```

or using `openssl`:

```
$ openssl rand -base64 32
```

Then add the generated *setKey()* line to your dnsdist configuration file, along with a *controlSocket()*:

```
controlSocket('192.0.2.53:5199') -- Listen on this IP and port for client connections
setKey("ENCODED KEY")            -- Shared secret for the console
```

Now you can run `dnsdist -c` to connect to the console. This makes dnsdist read its configuration file and use the *controlSocket()* and *setKey()* statements to set up its connection to the server.

If you want to connect over the network, create a configuration file with the same two statements and run `dnsdist -C /path/to/configfile -c`.

Alternatively, you can specify the address and key on the client commandline:

```
dnsdist -k "ENCODED KEY" -c 192.0.2.53:5199
```

Warning: This will leak the key into your shell's history and is **not** recommended.

Since 1.3.0, dnsdist supports restricting which client can connect to the console with an ACL:

```
controlSocket('192.0.2.53:5199')
setConsoleACL('192.0.2.0/24')
```

The default value is '127.0.0.1', restricting the use of the console to local users. Please make sure that encryption is enabled before using *addConsoleACL()* or *setConsoleACL()* to allow connection from remote clients. Even if the console is restricted to local users, the use of encryption is still strongly advised to prevent unauthorized local users from connecting to the console.

CHAPTER

TEN

DNS-OVER-HTTP/3 (DOH3)

Note: This guide is about DNS over HTTP/3. For DNS over HTTP/1 and DNS over HTTP/2, please see *DNS-over-HTTPS (DoH)*

dnsdist supports DNS-over-HTTP/3 (DoH3) for incoming queries since 1.9.0. To see if the installation supports this, run `dnsdist --version`. If the output shows `dns-over-http3` incoming DNS-over-HTTP/3 is supported.

10.1 Incoming

Adding a listen port for DNS-over-HTTP/3 can be done with the *`addDOH3Local()`* function, e.g.:

```
addDOH3Local('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/
↪private/example.com.key')
```

This will make **dnsdist** listen on [2001:db8:1:f00::1]:443 on UDP, and will use the provided certificate and key to serve incoming DoH3 connections.

The fourth parameter, if present, indicates various options. For instance, you can change the congestion control algorithm used. An example is:

```
addDOH3Local('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/
↪private/example.com.key', {congestionControlAlgo="bbr"})
```

A particular attention should be taken to the permissions of the certificate and key files. Many ACME clients used to get and renew certificates, like CertBot, set permissions assuming that services are started as root, which is no longer true for dnsdist as of 1.5.0. For that particular case, making a copy of the necessary files in the /etc/dnsdist directory is advised, using for example CertBot's `--deploy-hook` feature to copy the files with the right permissions after a renewal.

More information about sessions management can also be found in *TLS Sessions Management*.

10.1.1 Advertising DNS over HTTP/3 support

If DNS over HTTP/2 is also enabled in the configuration via *`addDOHLocal()`* (see *DNS-over-HTTPS (DoH)* for more information), it might be useful to advertise DNS over HTTP/3 support via the `Alt-Svc` header:

```
addDOHLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/
↪private/example.com.key', "/dns", {customResponseHeaders={["alt-svc"]="h3=\":443\
↪""}})
```

This will advertise that HTTP/3 is available on the same IP, port UDP/443.

CHAPTER

ELEVEN

DNS-OVER-HTTPS (DOH)

Note: This guide is about DNS over HTTP/1 and DNS over HTTP/2. For DNS over HTTP/3, please see *DNS-over-HTTP/3 (DoH3)*

dnsdist supports DNS-over-HTTPS (DoH, standardized in RFC 8484) for incoming queries since 1.4.0, and for outgoing queries since 1.7.0. To see if the installation supports this, run `dnsdist --version`. If the output shows `dns-over-https(DOH)` (`dns-over-https(h2o nghttp2)`, `dns-over-https(h2o)` or `dns-over-https(nghttp2)` since 1.9.0) , incoming DNS-over-HTTPS is supported. If `outgoing-dns-over-https(nghttp2)` shows up then outgoing DNS-over-HTTPS is supported.

11.1 Incoming

Adding a listen port for DNS-over-HTTPS can be done with the `addDOHLocal()` function, e.g.:

```
addDOHLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/
↪private/example.com.key')
```

This will make **dnsdist** listen on [2001:db8:1:f00::1]:443 on TCP, and will use the provided certificate and key to serve incoming TLS connections.

In order to support multiple certificates and keys, for example an ECDSA and an RSA one, the following syntax may be used instead:

```
addDOHLocal('2001:db8:1:f00::1', {'/etc/ssl/certs/example.com.rsa.pem', '/etc/ssl/
↪certs/example.com.ecdsa.pem'}, {'/etc/ssl/private/example.com.rsa.key', '/etc/
↪ssl/private/example.com.ecdsa.key'})
```

The certificate chain presented by the server to an incoming client will then be selected based on the algorithms this client advertised support for.

A fourth parameter may be added to specify the URL path(s) used by DoH. If you want your DoH server to handle `https://example.com/dns-query-endpoint`, you have to add `"/dns-query-endpoint"` to the call to `addDOHLocal()`. It is optional and defaults to `/` in 1.4.0, and `/dns-query` since 1.5.0.

The fifth parameter, if present, indicates various options. For instance, you use it to indicate custom HTTP headers. An example is:

```
addDOHLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/
↪private/example.com.key', "/dns", {customResponseHeaders={["x-foo"]="bar"}})
```

A more complicated (and more realistic) example is when you want to indicate metainformation about the server, such as the stated policy (privacy statement and so on). We use the link types of RFC 8631:

```
addDOHLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/
↪private/example.com.key', "/", {customResponseHeaders={["link"]="<https://
↪example.com/policy.html> rel=\\"service-meta\\"; type=\\"text/html\\""}})
```

(continues on next page)

(continued from previous page)

A particular attention should be taken to the permissions of the certificate and key files. Many ACME clients used to get and renew certificates, like CertBot, set permissions assuming that services are started as root, which is no longer true for dnsdist as of 1.5.0. For that particular case, making a copy of the necessary files in the /etc/dnsdist directory is advised, using for example CertBot's `--deploy-hook` feature to copy the files with the right permissions after a renewal.

More information about sessions management can also be found in *TLS Sessions Management*.

11.1.1 Advertising DNS over HTTP/3 support

If DNS over HTTP/3 is also enabled in the configuration via `addDOH3Local()` (see *DNS-over-HTTP/3 (DoH3)* for more information), it might be useful to advertise this support via the `Alt-Svc` header:

```
addDOHLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/private/example.com.key', "/dns", {customResponseHeaders={["alt-svc"]="h3=\":443\""}})
```

This will advertise that HTTP/3 is available on the same IP, port UDP/443.

11.1.2 Custom responses

It is also possible to set HTTP response rules to intercept HTTP queries early, before the DNS payload, if any, has been processed, to send custom responses including error pages, redirects or even serve static content. First a rule needs to be defined using `newDOHResponseMapEntry()`, then a set of rules can be applied to a DoH frontend via `DOHFrontend:setResponsesMap()`. For example, to send an HTTP redirect to queries asking for `/rfc`, the following configuration can be used:

```
map = { newDOHResponseMapEntry("^/rfc$", 307, "https://www.rfc-editor.org/info/rfc8484") }
dohFE = getDOHFrontend(0)
dohFE:setResponsesMap(map)
```

11.1.3 DNS over HTTP

In case you want to run DNS-over-HTTPS behind a reverse proxy you probably don't want to encrypt your traffic between reverse proxy and dnsdist. To let dnsdist listen for DoH queries over HTTP on localhost at port 8053 add one of the following to your config:

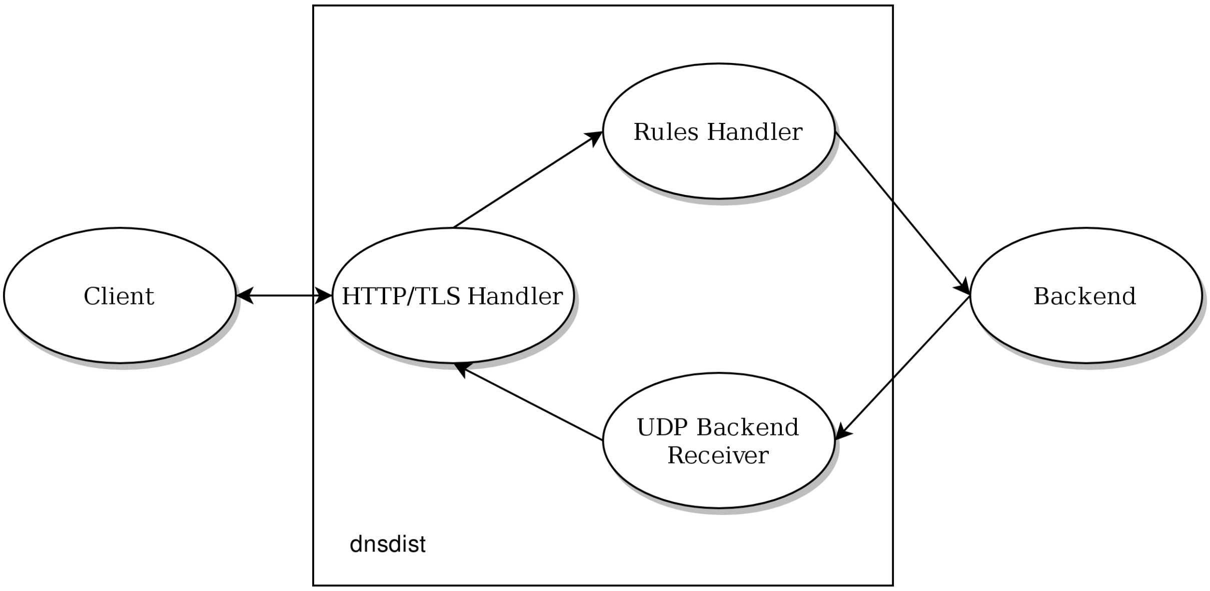
```
addDOHLocal("127.0.0.1:8053")
addDOHLocal("127.0.0.1:8053", nil, nil, "/", { reusePort=true })
```

11.1.4 HTTP/1 support

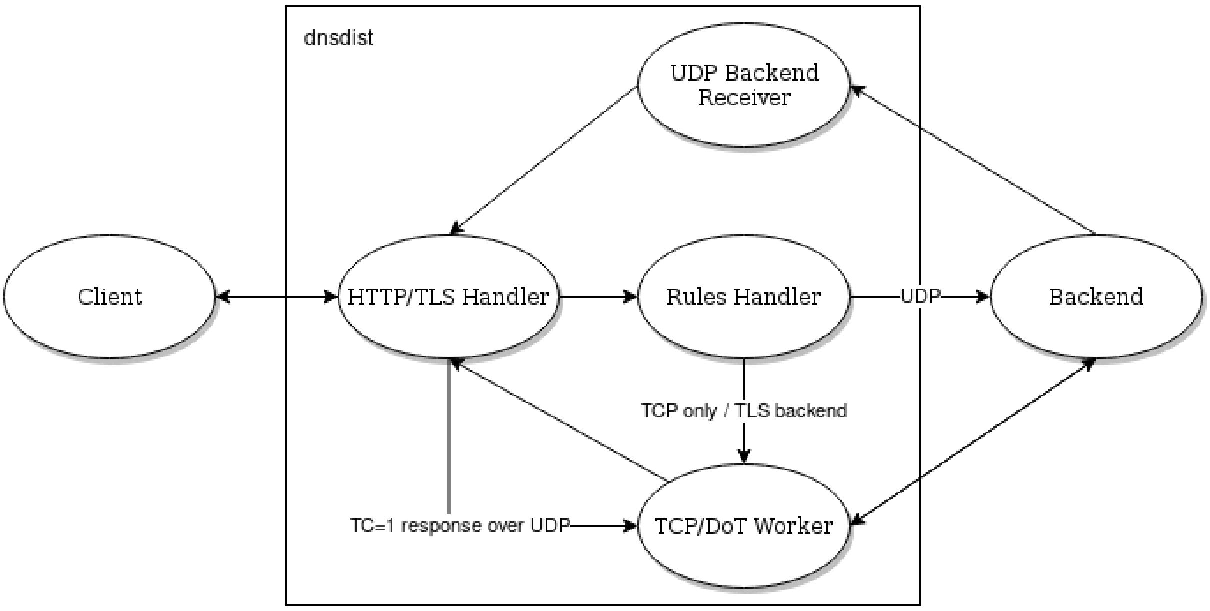
dnsdist initially relied on the `h2o` library to support incoming DNS over HTTPS. Since 1.9.0, `h2o` has been deprecated and `nghttp2` is the preferred library for incoming DoH support, because `h2o` has unfortunately really never been maintained in a way that is suitable for use as a library (see https://github.com/h2o/h2o/issues/3230). While we took great care to make the migration as painless as possible, `h2o` supported HTTP/1 while `nghttp2` does not. This is not an issue for actual DNS over HTTPS clients that support HTTP/2, but might be one in setups running dnsdist behind a reverse-proxy that does not support HTTP/2, like nginx. We do not plan on implementing HTTP/1, and recommend using HTTP/2 between the reverse-proxy and dnsdist for performance reasons. For nginx in particular, a possible work-around is to use the grpc_pass directive as suggested in their bug tracker.

11.1.5 Internal design

The internal design used for DoH handling uses two threads per `addDOHLocal()` directive. The first thread will handle the HTTP/2 communication with the client and pass the received DNS queries to a second thread which will apply the rules and pass the query to a backend, over **UDP** (except if the backend is TCP-only, or uses DNS over TLS, see the second schema below). The response will be received by the regular UDP response handler for that backend and passed back to the first thread. That allows the first thread to be low-latency dealing with TLS and HTTP/2 only and never blocking.



The fact that the queries are forwarded over UDP means that a large UDP payload size should be configured between dnsdist and the backend to avoid most truncation issues, and dnsdist will advise a 4096-byte UDP Payload Buffer size. UDP datagrams can still be larger than the MTU as long as fragmented datagrams are not dropped on the path between dnsdist and the backend. Since 1.7.0, truncated answers received over UDP for a DoH query will lead to a retry over TCP, passing the query to a TCP worker, as illustrated below.



11.1.6 Investigating issues

dnsdist provides a lot of counters to investigate issues:

- `showTCPStats()` will display a lot of information about current and passed connections
- `showTLSErrorCounters()` some metrics about why TLS sessions failed to establish
- `showDOHResponseCodes()` returns metrics about HTTP response codes sent by dnsdist

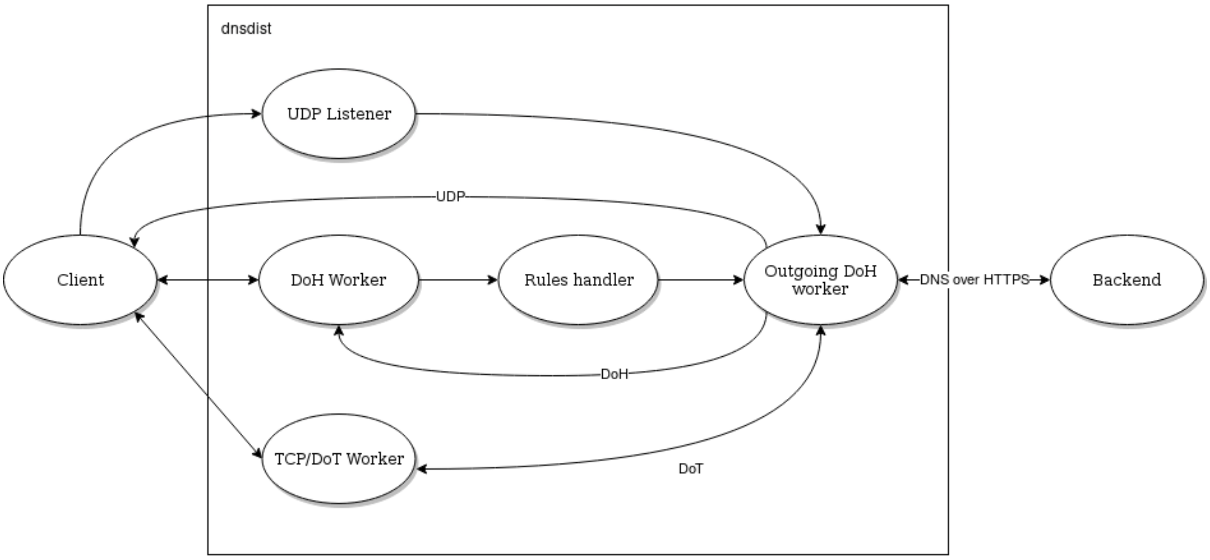
11.2 Outgoing

Support for securing the exchanges between dnsdist and the backend will be implemented in 1.7.0, and will lead to all queries, regardless of whether they were initially received by dnsdist over UDP, TCP, DoT or DoH, being forwarded over a secure DNS over HTTPS channel. That support can be enabled via the `dohPath` parameter of the `newServer()` command. Additional parameters control the TLS provider used (`tls`), the validation of the certificate presented by the backend (`caStore`, `validateCertificates`), the actual TLS ciphers used (`ciphers`, `ciphersTLS13`) and the SNI value sent (`subjectName`).

```
newServer({address="[2001:DB8::1]:443", tls="openssl", subjectName="doh.powerdns.
↪com", dohPath="/dns-query", validateCertificates=true})
```

11.2.1 Internal design

The incoming queries, after the processing of rules if any, are passed to one of the DoH workers over a pipe. The DoH worker handles the communication with the backend, retrieves the response, and either responds directly to the client (queries coming over UDP) or pass it back over a pipe to the initial thread (queries coming over TCP, DoT or DoH). The number of outgoing DoH worker threads can be configured using `setOutgoingDoHWorkerThreads()`.



CHAPTER TWELVE

DNS-OVER-QUIC (DOQ)

dnsdist supports DNS-over-QUIC (DoQ, standardized in RFC 9250) for incoming queries since 1.9.0. To see if the installation supports this, run `dnsdist --version`. If the output shows `dns-over-quic` incoming DNS-over-QUIC is supported.

12.1 Incoming

Adding a listen port for DNS-over-QUIC can be done with the *addDOQLocal()* function, e.g.:

```
addDOQLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/
↪private/example.com.key')
```

This will make **dnsdist** listen on [2001:db8:1:f00::1]:853 on UDP, and will use the provided certificate and key to serve incoming DoQ connections.

The fourth parameter, if present, indicates various options. For instance, you can change the congestion control algorithm used. An example is:

```
addDOQLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/
↪private/example.com.key', {congestionControlAlgo="bbr"})
```

A particular attention should be taken to the permissions of the certificate and key files. Many ACME clients used to get and renew certificates, like CertBot, set permissions assuming that services are started as root, which is no longer true for dnsdist as of 1.5.0. For that particular case, making a copy of the necessary files in the /etc/dnsdist directory is advised, using for example CertBot's `--deploy-hook` feature to copy the files with the right permissions after a renewal.

More information about sessions management can also be found in *TLS Sessions Management*.

CHAPTER

THIRTEEN

DNS-OVER-TLS

13.1 Incoming

Since version 1.3.0, **dnsdist** supports DNS-over-TLS for incoming queries. To see if the installation supports this, run `dnsdist --version`. If the output shows `dns-over-tls` with one or more SSL libraries in brackets, DNS-over-TLS is supported.

Adding a listen port for DNS-over-TLS can be done with the *addTLSLocal()* function, e.g.:

```
addTLSLocal('192.0.2.55', '/etc/ssl/certs/example.com.pem', '/etc/ssl/private/
↪example.com.key')
```

This will make **dnsdist** listen on 192.0.2.55:853 on TCP, and will use the provided certificate and key to serve incoming TLS connections.

In order to support multiple certificates and keys, for example an ECDSA and an RSA one, the following syntax may be used instead:

```
addTLSLocal('192.0.2.55', {'/etc/ssl/certs/example.com.rsa.pem', '/etc/ssl/certs/
↪example.com.ecdsa.pem'}, {'/etc/ssl/private/example.com.rsa.key', '/etc/ssl/
↪private/example.com.ecdsa.key'})
```

The certificate chain presented by the server to an incoming client will then be selected based on the algorithms this client advertised support for.

A particular attention should be taken to the permissions of the certificate and key files. Many ACME clients used to get and renew certificates, like CertBot, set permissions assuming that services are started as root, which is no longer true for dnsdist as of 1.5.0. For that particular case, making a copy of the necessary files in the /etc/dnsdist directory is advised, using for example CertBot's `--deploy-hook` feature to copy the files with the right permissions after a renewal.

More information about sessions management can also be found in *TLS Sessions Management*.

13.2 Outgoing

Support for securing the exchanges between dnsdist and the backend will be implemented in 1.7.0, and will lead to all queries, regardless of whether they were initially received by dnsdist over UDP, TCP, DoT or DoH, being forwarded over a secure DNS over TLS channel. That support can be enabled via the `tls` parameter of the *newServer()* command. Additional parameters control the validation of the certificate presented by the backend (`caStore`, `validateCertificates`), the actual TLS ciphers used (`ciphers`, `ciphersTLS13`) and the SNI value sent (`subjectName`).

```
newServer({address="[2001:DB8::1]:853", tls="openssl", subjectName="dot.powerdns.
↪com", validateCertificates=true})
```

13.3 Investigating issues

dnsdist provides a lot of counters to investigate issues:

- `showTCPStats()` will display a lot of information about current and passed connections
- `showTLSErrorCounters()` some metrics about why TLS sessions failed to establish

CHAPTER

FOURTEEN

DNSCRYPT

dnsdist, when compiled with `--enable-dnscrypt`, can be used as a DNSCrypt server, uncurving queries before forwarding them to downstream servers and curving responses back. To make **dnsdist** listen to incoming DNSCrypt queries on 127.0.0.1 port 8443, with a provider name of “2.providername”, using a resolver certificate and associated key stored respectively in the resolver.cert and resolver.key files, the *addDNSCryptBind()* directive can be used:

```
addDNSCryptBind("127.0.0.1:8443", "2.providername", "/path/to/resolver.cert", "/
↪path/to/resolver.key")
```

To generate the provider and resolver certificates and keys, you can simply do:

```
> generateDNSCryptProviderKeys("/path/to/providerPublic.key", "/path/to/
↪providerPrivate.key")
Provider fingerprint is:␣
↪E1D7:2108:9A59:BF8D:F101:16FA:ED5E:EA6A:9F6C:C78F:7F91:AF6B:027E:62F4:69C3:B1AA
> generateDNSCryptCertificate("/path/to/providerPrivate.key", "/path/to/resolver.
↪cert", "/path/to/resolver.key", serial, validFrom, validUntil)
```

Ideally, the certificates and keys should be generated on an offline dedicated hardware and not on the resolver. The resolver key should be regularly rotated and should never touch persistent storage, being stored in a tmpfs with no swap configured.

You can display the currently configured DNSCrypt binds with:

```
> showDNSCryptBinds()
#   Address              Provider Name        Serial   Validity              P.␣
↪Serial P. Validity
0   127.0.0.1:8443       2.name               14       2016-04-10 08:14:15   0    ␣
↪     -
```

If you forgot to write down the provider fingerprint value after generating the provider keys, you can use *printDNSCryptProviderFingerprint()* to retrieve it later:

```
> printDNSCryptProviderFingerprint("/path/to/providerPublic.key")
Provider fingerprint is:␣
↪E1D7:2108:9A59:BF8D:F101:16FA:ED5E:EA6A:9F6C:C78F:7F91:AF6B:027E:62F4:69C3:B1A
```

CHAPTER

FIFTEEN

CONFIGURING DOWNSTREAM SERVERS

As dnsdist is a loadbalancer and does not do any DNS resolving or serving by itself, it needs downstream servers. To add downstream servers, either include them on the command line:

```
dnsdist -l 130.161.252.29 -a 130.161.0.0/16 8.8.8.8 208.67.222.222 2620:0:ccc::2␣
↪2620:0:ccd::2
```

Or add them to the configuration file:

```
setLocal("130.161.252.29:53")
setACL("130.161.0.0/16")
newServer("8.8.8.8")
newServer("208.67.222.222")
newServer("2620:0:ccc::2")
newServer("2620:0:0ccd::2")
```

These two equivalent configurations give you sane load balancing using a very sensible distribution policy. Many users will simply be done with this configuration. It works as well for authoritative as for recursive servers.

15.1 Healthcheck

dnsdist uses health-check queries, sent once every second, to determine the availability of a backend server. Since 1.8.0, it also supports a `lazy` health-checking mode which only sends active health-check queries after a configurable threshold of regular queries have failed, see below.

By default, an `A` query for the "a.root-servers.net." name is sent. A different query type, class and target can be specified by passing, respectively, the `checkType`, `checkClass` and `checkName` parameters to *`newServer()`*. The interval between two health-check queries can be set via the `checkInterval` interval parameter, and the amount of time for a response to be received via the `checkTimeout` one.

Since the 1.3.0 release, the `checkFunction` option is also supported, taking a `Lua` function as parameter. This function receives a DNSName, two integers and a `DNSHeader` object (*DNSHeader (dh) object*) representing the QName, QType and QClass of the health check query as well as the DNS header, as they are defined before the function was called. The function must return a DNSName and two integers representing the new QName, QType and QClass, and can directly modify the `DNSHeader` object.

The following example sets the CD flag to true and change the QName to "powerdns.com." and the QType to AAAA while keeping the initial QClass.

```
function myHealthCheck(qname, qtype, qclass, dh)
  dh:setCD(true)

  return newDNSName("powerdns.com."), DNSQType.AAAA, qclass
end

newServer({address="2620:0:0ccd::2", checkFunction=myHealthCheck})
```

The default behavior is to consider any valid response with an RCODE different from ServFail as valid. If the mustResolve parameter of *newServer()* is set to true, a response will only be considered valid if its RCODE differs from NXDomain, ServFail and Refused.

The number of health check failures before a server is considered down is configurable via the maxCheckFailures parameter, defaulting to 1. In the same way, the number of consecutive successful health checks needed for a server to be considered available can be set via the rise parameter, defaulting to 1.

The CD flag can be set on the query by setting setCD to true. e.g.:

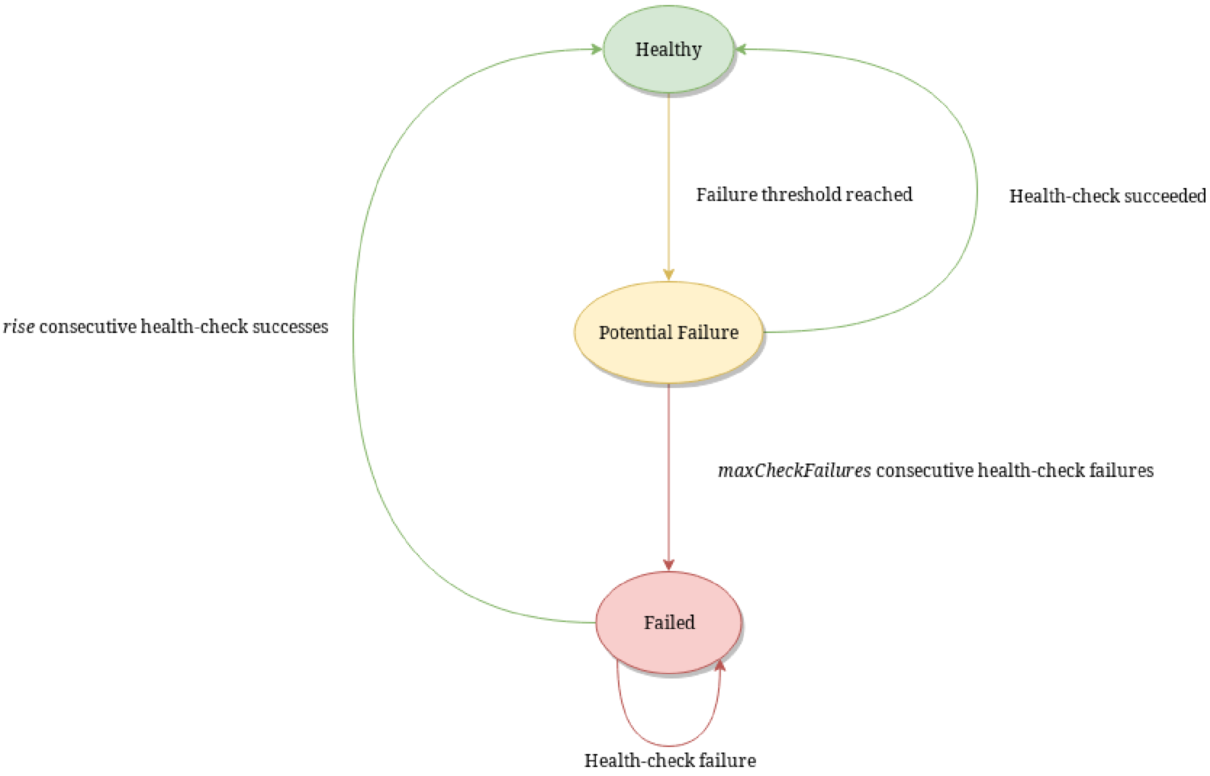
```
newServer({address="192.0.2.1", checkType="AAAA", checkClass=DNSClass.CHAOS,␣
↪checkName="a.root-servers.net.", mustResolve=true})
```

You can turn on logging of health check errors using the *setVerboseHealthChecks()* function.

15.1.1 Lazy health-checking

In some setups, especially on low-end devices, it might not make sense to actively send queries to the backend at a regular interval. Using the feedback from the results of regular queries can instead be used to infer if a backend might not be working properly.

Since 1.8.0, dnsdist implements a lazy mode that can be set via the healthCheckMode option on *newServer()*. In this mode, dnsdist will only send active health-check queries after seeing a configurable amount of regular queries failing. It will then place the backend in a PotentialFailure state, from the initial Healthy one, and send health-check queries every checkInterval seconds. If maxCheckFailures of these fail, the backend is then moved to a Failed state and marked as down, and active health-check queries are sent every lazyHealthCheckFailedInterval seconds. After rise successful, consecutive queries, the backend will be moved back to the Healthy state and marked as up again, and health-check queries will stop.


Healthy
Failure threshold reached
Health-check succeeded
rise consecutive health-check successes
Potential Failure
maxCheckFailures consecutive health-check failures
Failed
Health-check failure

The threshold of failed regular queries is configured via lazyHealthCheckThreshold, indicating of percentage of regular queries that should have resulted in a failure over the last recent queries. Only the results of the last lazyHealthCheckSampleSize queries will be considered, as the results are kept in a in-memory circular buffer. The results of at least lazyHealthCheckMinSampleCount queries should be present for the threshold to be considered meaningful, to avoid an issue with a too small sample.

By default both queries that resulted in a timeout and those that received a `ServFail` answer are considered failures, but it is possible to set `lazyHealthCheckMode` to `TimeoutOnly` so that only timeouts are considered failures.

So for example, if we set `healthCheckMode` to `lazy`, `lazyHealthCheckSampleSize` to 100, `lazyHealthCheckMinSampleCount` to 10, `lazyHealthCheckThreshold` to 30, `maxCheckFailures` to 2 and `rise` to 2:

- nothing will happen until at least 10 queries have been received
- only the results of the last 100 queries will be considered
- if at least 30 of these last 100 have failed, the threshold will be reached and active health-check queries will be sent every `checkInterval` seconds
- if the health-check query is successful, the backend will stay `up` and no more query will be sent
- but if instead two consecutive queries fail, the backend will be marked as `down` and health-check queries will be sent every `lazyHealthCheckFailedInterval` seconds
- it will take two consecutive, successful health-checks for the backend to go back to `Healthy` and be marked *up* again

```
newServer({address="192.0.2.1", healthCheckMode='lazy', checkInterval=1,␣
↪lazyHealthCheckFailedInterval=30, rise=2, maxCheckFailures=3,␣
↪lazyHealthCheckThreshold=30, lazyHealthCheckSampleSize=100, ␣
↪lazyHealthCheckMinSampleCount=10, lazyHealthCheckMode='TimeoutOnly'})
```

The 'lazy' mode also supports using an exponential back-off time between health-check queries, once a backend has been moved to the 'down' state. This can be enabled by setting the `lazyHealthCheckUseExponentialBackOff` parameter to 'true'. Once the backend has been marked as 'down', the first query will be sent after `lazyHealthCheckFailedInterval` seconds, the second one after 2 times `lazyHealthCheckFailedInterval` seconds, the third after 4 times `lazyHealthCheckFailedInterval` seconds, and so on and so forth, until `lazyHealthCheckMaxBackOff` has been reached. Then probes will be sent every `lazyHealthCheckMaxBackOff` seconds (default is 3600 so one hour) until the backend comes 'up' again.

15.2 Source address selection

In multi-homed setups, it can be useful to be able to select the source address or the outgoing interface used by dnsdist to contact a downstream server. This can be done by using the *source* parameter:

```
newServer({address="192.0.2.1", source="192.0.2.127"})
newServer({address="192.0.2.1", source="eth1"})
newServer({address="192.0.2.1", source="192.0.2.127@eth1"})
```

The supported values for source are:

- an IPv4 or IPv6 address, which must exist on the system
- an interface name
- an IPv4 or IPv6 address followed by '@' then an interface name

Please note that specifying the interface name is only supported on system having *IP_PKTINFO*.

15.3 Securing the channel

15.4 Securing the path to the backend

As explained briefly in the quickstart guide, dnsdist has always been designed as a load-balancer placed in front of authoritative or recursive servers, assuming that the network path between dnsdist and these servers is trusted. This is particularly important because for performance reasons it uses a single connected socket for UDP exchanges by default, and easy to predict DNS query IDs, which makes it easy for an attacker to poison responses.

If dnsdist is instead intended to be deployed in such a way that the path to its backend is not secure, the UDP protocol should not be used, and 'TCP-only', DNS over TLS and DNS over HTTPS protocols used instead, as supported since 1.7.0.

Using these protocols leads to all queries, regardless of whether they were initially received by dnsdist over UDP, TCP, DoT or DoH, being forwarded over a TCP socket, a secure DNS over TLS channel or a secure DNS over HTTPS channel.

The TCP-only mode for a backend can be enabled by using the `tcpOnly` parameter of the *`newServer()`* command.

The DNS over TLS mode via the `tls` parameter of the *`newServer()`* command. Additional parameters control the validation of the certificate presented by the backend (`caStore`, `validateCertificates`), the actual TLS ciphers used (`ciphers`, `ciphersTLS13`) and the SNI value sent (`subjectName`).

The DNS over HTTPS mode in the same way than DNS over TLS but with the additional `dohPath` keyword indicating that DNS over HTTPS should be used instead of DNS over TLS.

If it is absolutely necessary to support UDP exchanges over an untrusted network, a few options have been introduced in 1.8.0 to make spoofing attempts harder:

- *`setRandomizedIdsOverUDP()`* will randomize the IDs in outgoing queries, at a small performance cost. *`setMaxUDPOutstanding()`* should be set at its highest possible value (default since 1.4.0) to make that setting fully efficient.
- *`setRandomizedOutgoingSockets()`* can be used to randomize the outgoing socket used when forwarding a query to a backend. This requires configuring the backend to use more than one outgoing socket via the `sockets` parameter of *`newServer()`* to be of any use.

CHAPTER

SIXTEEN

DYNAMIC RULE GENERATION

Dynamic Blocks can be seen are short-lived rules, automatically inserted based on configurable thresholds and the analysis of recently received traffic, and automatically removed after a configurable amount of time.

The analyzed traffic is the one kept by dnsdist in its in-memory ring buffers. The number of entries kept in these ring buffers can be set via the `setRingBuffersSize()` directive, and the impact in terms of CPU and memory consumption is described in *Performance Tuning*.

That number of entries is crucial for the rate-based rules, like `DynBlockRulesGroup:setQueryRate()`, as they will never match if the number of entries in the ring buffer is too small for the required rate, as explained in more details below.

To set dynamic rules, based on recent traffic, define a function called `maintenance()` in Lua. It will get called every second, and from this function you can set rules to block traffic based on statistics. More exactly, the thread handling the `maintenance()` function will sleep for one second between each invocation, so if the function takes several seconds to complete it will not be invoked exactly every second.

As an example:

```
local dbr = dynBlockRulesGroup()
dbr:setQueryRate(20, 10, "Exceeded query rate", 60)

function maintenance()
  dbr:apply()
end
```

This will dynamically block all hosts that exceeded 20 queries/s as measured over the past 10 seconds, and the dynamic block will last for 60 seconds.

DynBlockRulesGroup is a very efficient way of processing dynamic blocks that was introduced in 1.3.0. Before that, it was possible to use `addDynBlocks()` instead:

```
-- this is a legacy method, please see above for DNSdist >= 1.3.0
function maintenance()
    addDynBlocks(exceedQRate(20, 10), "Exceeded query rate", 60)
end
```

Dynamic blocks in force are displayed with `showDynBlocks()` and can be cleared with `clearDynBlocks()`. They return a table whose key is a `ComboAddress` object, representing the client's source address, and whose value is an integer representing the number of queries matching the corresponding condition (for example the qtype for `exceedQTypeRate()`, rcode for `exceedServFails()`).

All exceed-functions are documented in the *Configuration Reference*.

Dynamic blocks drop matched queries by default, but this behavior can be changed with `setDynBlocksAction()`. For example, to send a REFUSED code instead of dropping the query:

```
setDynBlocksAction(DNSAction.Refused)
```

Please see the documentation for `setDynBlocksAction()` to confirm which actions are supported.

16.1 DynBlockRulesGroup

Starting with dnsdist 1.3.0, a new `dynBlockRulesGroup()` function can be used to return a `DynBlockRulesGroup` instance, designed to make the processing of multiple rate-limiting rules faster by walking the query and response buffers only once for each invocation, instead of once per existing *exceed*()* invocation.

The new syntax would be:

```
local dbr = dynBlockRulesGroup()
dbr:setQueryRate(30, 10, "Exceeded query rate", 60)
dbr:setRCodeRate(DNSRCode.NXDOMAIN, 20, 10, "Exceeded NXD rate", 60)
dbr:setRCodeRate(DNSRCode.SERVFAIL, 20, 10, "Exceeded ServFail rate", 60)
dbr:setQTypeRate(DNSQType.ANY, 5, 10, "Exceeded ANY rate", 60)
dbr:setResponseByteRate(10000, 10, "Exceeded resp BW rate", 60)

function maintenance()
  dbr:apply()
end
```

Before 1.3.0 the legacy syntax was:

```
function maintenance()
  -- this example is using legacy methods, please see above for DNSdist >= 1.3.0
  addDynBlocks(exceedQRate(30, 10), "Exceeded query rate", 60)
  addDynBlocks(exceedNXDOMAINs(20, 10), "Exceeded NXD rate", 60)
  addDynBlocks(exceedServFails(20, 10), "Exceeded ServFail rate", 60)
  addDynBlocks(exceedQTypeRate(DNSQType.ANY, 5, 10), "Exceeded ANY rate", 60)
  addDynBlocks(exceedRespByterate(1000000, 10), "Exceeded resp BW rate", 60)
end
```

The old syntax would walk the query buffer 2 times and the response one 3 times, while the new syntax does it only once for each. It also reuse the same internal table to keep track of the source IPs, reducing the CPU usage.

DynBlockRulesGroup also offers the ability to specify that some network ranges should be excluded from dynamic blocking:

```
-- do not add dynamic blocks for hosts in the 192.0.2.0/24 and 2001:db8::/32 ranges
dbr:excludeRange({"192.0.2.0/24", "2001:db8::/32" })
-- except for 192.0.2.1
dbr:includeRange("192.0.2.1/32")
```

Since 1.3.3, it's also possible to define a warning rate. When the query or response rate raises above the warning level but below the trigger level, a warning message will be issued along with a no-op block. If the rate reaches the trigger level, the regular action is applied.

```
local dbr = dynBlockRulesGroup()
-- Generate a warning if we detect a query rate above 100 qps for at least 10s.
-- If the query rate raises above 300 qps for 10 seconds, we'll block the client␣
↪for 60s.
dbr:setQueryRate(300, 10, "Exceeded query rate", 60, DNSAction.Drop, 100)
```

Since 1.6.0, if a default eBPF filter has been set via `setDefaultBPFFilter()` dnsdist will automatically try to use it when a "drop" dynamic block is inserted via a *DynBlockRulesGroup*. eBPF blocks are applied in kernel space and are much more efficient than user space ones. Note that a regular block is also inserted so that any failure will result in a regular block being used instead of the eBPF one.

16.2 Rate rules and size of the ring buffers

As explained in the introduction, the whole dynamic block feature is based on analyzing the recent traffic kept in dnsdist's in-memory ring buffers, whose content can be inspected via `grepq()`.

The sizing of the buffers, in addition to having performance impacts explained in *Performance Tuning*, directly impacts some of the dynamic block rules, like the rate and ratio-based ones.

For example, if `DynBlockRulesGroup:setQueryRate()` is used to request the blocking for 60s of any client exceeding 1000 qps over 10s, like this:

```
dbr:setQueryRate(1000, 10, "Exceeded query rate", 60, DNSAction.Drop)
```

For this rule to trigger, dnsdist will need to scan the ring buffers and find 1000 * 10 = 10000 queries, not older than 10s, from that client. Since a ring buffer has a fixed size, and new entries override the oldest ones when the buffer is full, that only works if there are enough entries in the buffer.

This is even more obvious for the ratio-based rules, when they have a minimum number of responses set, because in that case they clearly require that number of responses to fit in the buffer.

That requirement could be lifted a bit by the use of sampling, meaning that only one query out of 10 would be recorded, for example, and the total amount would be inferred from the queries present in the buffer. As of 1.7.0, sampling as unfortunately not been implemented yet.

CHAPTER

SEVENTEEN

GUIDES

These chapters contain several guides and nuggets of information regarding dnsdist operation and accomplishing specific goals.

17.1 Built-in webserver

To visually interact with dnsdist, try adding `webserver()` and `setWebserverConfig()` directives to the configuration:

```
webserver("127.0.0.1:8083")
setWebserverConfig({password="supersecretpassword", apiKey="supersecretAPIkey"})
```

Now point your browser at http://127.0.0.1:8083 and log in with any username, and that password. Enjoy!

Since 1.5.0, only connections from 127.0.0.1 and ::1 are allowed by default. To allow connections from 192.0.2.0/24 but not from 192.0.2.1, instead:

```
setWebserverConfig({password="supersecretpassword", apiKey="supersecretAPIkey",␣
↪acl="192.0.2.0/24, !192.0.2.1"})
```

17.1.1 Security of the Webserver

The built-in webserver serves its content from inside the binary, this means it will not and cannot read from disk.

By default, our web server sends some security-related headers:

```
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-Permitted-Cross-Domain-Policies: none
X-XSS-Protection: 1; mode=block
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-inline'
```

You can override those headers, or add custom headers by using the last parameter to `setWebserverConfig()`. For example, to remove the X-Frame-Options header and add a X-Custom one:

```
setWebserverConfig({password="supersecretpassword", apiKey="supersecretAPIkey",␣
↪customHeaders={["X-Frame-Options"]= "", ["X-Custom"]="custom"} })
```

Credentials can be changed at run time using the `setWebserverConfig()` function.

17.1.2 dnsdist API

To access the API, the *apikey* must be set in the `setWebserverConfig()` function. Use the API, this key will need to be sent to dnsdist in the `X-API-Key` request header. An HTTP 401 response is returned when a wrong or no API key is received. A 404 response is generated is the requested endpoint does not exist. And a 405 response is returned when the HTTP method is not allowed.

URL Endpoints

GET /jsonstat

Get statistics from dnsdist in JSON format. The `Accept` request header is ignored. This endpoint accepts a `command` query for different statistics:

- `stats`: Get all *Statistics* as a JSON dict
- `dynblocklist`: Get all current *dynamic blocks*, keyed by netmask
- `ebpfblocklist`: Idem, but for *eBPF* blocks

Example request:

```
GET /jsonstat?command=stats HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Connection: close
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-
↪inline'
Content-Type: application/json
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 1; mode=block

{"acl-drops": 0, "cache-hits": 0, "cache-misses": 0, "cpu-sys-msec": 633,
↪"cpu-user-msec": 499, "downstream-send-errors": 0, "downstream-timeouts
↪": 0, "dyn-block-nmg-size": 1, "dyn-blocked": 3, "empty-queries": 0, "fd-
↪usage": 17, "latency-avg100": 7651.3982737482893, "latency-avg1000": 860.
↪05142763680249, "latency-avg10000": 87.032142373878372, "latency-
↪avg1000000": 0.87146026426551759, "latency-slow": 0, "latency0-1": 0,
↪"latency1-10": 0, "latency10-50": 22, "latency100-1000": 1, "latency50-
↪100": 0, "no-policy": 0, "noncompliant-queries": 0, "noncompliant-
↪responses": 0, "over-capacity-drops": 0, "packetcache-hits": 0,
↪"packetcache-misses": 0, "queries": 26, "rdqueries": 26, "real-memory-
↪usage": 6078464, "responses": 23, "rule-drop": 0, "rule-nxdomain": 0,
↪"rule-refused": 0, "self-answered": 0, "server-policy": "leastOutstanding
↪", "servfail-responses": 0, "too-old-drops": 0, "trunc-failures": 0,
↪"uptime": 412}
```

Example request:

```
GET /jsonstat?command=dynblocklist HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

Example response:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Connection: close
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-
↪inline'
```

(continues on next page)

(continued from previous page)

```
Content-Type: application/json
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 1; mode=block

{"127.0.0.1/32": {"blocks": 3, "reason": "Exceeded query rate", "seconds":␣
↪10}}
```

Query Parameters

- **command** – one of `stats`, `dynblocklist` or `ebpfblocklist`

GET /metrics

Get statistics from dnsdist in Prometheus format.

Example request:

```
GET /metrics HTTP/1.1
```

Example response:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-
↪inline'
Content-Type: text/plain
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 1; mode=block

# HELP dnsdist_responses Number of responses received from backends
# TYPE dnsdist_responses counter
dnsdist_responses 0
# HELP dnsdist_servfail_responses Number of SERVFAIL answers received from␣
↪backends
# TYPE dnsdist_servfail_responses counter
dnsdist_servfail_responses 0
# HELP dnsdist_queries Number of received queries
# TYPE dnsdist_queries counter
dnsdist_queries 0
# HELP dnsdist_frontend_nxdomain Number of NXDomain answers sent to clients
# TYPE dnsdist_frontend_nxdomain counter
dnsdist_frontend_nxdomain 0
# HELP dnsdist_frontend_servfail Number of SERVFAIL answers sent to clients
# TYPE dnsdist_frontend_servfail counter
dnsdist_frontend_servfail 0
# HELP dnsdist_frontend_noerror Number of NoError answers sent to clients
# TYPE dnsdist_frontend_noerror counter
dnsdist_frontend_noerror 0
# HELP dnsdist_acl_drops Number of packets dropped because of the ACL
# TYPE dnsdist_acl_drops counter
dnsdist_acl_drops 0
# HELP dnsdist_rule_drop Number of queries dropped because of a rule
# TYPE dnsdist_rule_drop counter
dnsdist_rule_drop 0
# HELP dnsdist_rule_nxdomain Number of NXDomain answers returned because␣
↪of a rule
# TYPE dnsdist_rule_nxdomain counter
```

(continues on next page)

(continued from previous page)

```
dnsdist_rule_nxdomain 0
# HELP dnsdist_rule_refused Number of Refused answers returned because of␣
↪a rule
# TYPE dnsdist_rule_refused counter
dnsdist_rule_refused 0
# HELP dnsdist_rule_servfail Number of SERVFAIL answers received because␣
↪of a rule
# TYPE dnsdist_rule_servfail counter
dnsdist_rule_servfail 0
# HELP dnsdist_rule_truncated Number of truncated answers returned because␣
↪of a rule
# TYPE dnsdist_rule_truncated counter
dnsdist_rule_truncated 0
# HELP dnsdist_self_answered Number of self-answered responses
# TYPE dnsdist_self_answered counter
dnsdist_self_answered 0
# HELP dnsdist_downstream_timeouts Number of queries not answered in time␣
↪by a backend
# TYPE dnsdist_downstream_timeouts counter
dnsdist_downstream_timeouts 0
# HELP dnsdist_downstream_send_errors Number of errors when sending a␣
↪query to a backend
# TYPE dnsdist_downstream_send_errors counter
dnsdist_downstream_send_errors 0
# HELP dnsdist_trunc_failures Number of errors encountered while␣
↪truncating an answer
# TYPE dnsdist_trunc_failures counter
dnsdist_trunc_failures 0
# HELP dnsdist_no_policy Number of queries dropped because no server was␣
↪available
# TYPE dnsdist_no_policy counter
dnsdist_no_policy 0
# HELP dnsdist_latency0_1 Number of queries answered in less than 1ms
# TYPE dnsdist_latency0_1 counter
dnsdist_latency0_1 0
# HELP dnsdist_latency1_10 Number of queries answered in 1-10 ms
# TYPE dnsdist_latency1_10 counter
dnsdist_latency1_10 0
# HELP dnsdist_latency10_50 Number of queries answered in 10-50 ms
# TYPE dnsdist_latency10_50 counter
dnsdist_latency10_50 0
# HELP dnsdist_latency50_100 Number of queries answered in 50-100 ms
# TYPE dnsdist_latency50_100 counter
dnsdist_latency50_100 0
# HELP dnsdist_latency100_1000 Number of queries answered in 100-1000 ms
# TYPE dnsdist_latency100_1000 counter
dnsdist_latency100_1000 0
# HELP dnsdist_latency_slow Number of queries answered in more than 1␣
↪second
# TYPE dnsdist_latency_slow counter
dnsdist_latency_slow 0
# HELP dnsdist_latency_avg100 Average response latency in microseconds of␣
↪the last 100 packets
# TYPE dnsdist_latency_avg100 gauge
dnsdist_latency_avg100 0
# HELP dnsdist_latency_avg1000 Average response latency in microseconds of␣
↪the last 1000 packets
# TYPE dnsdist_latency_avg1000 gauge
dnsdist_latency_avg1000 0
# HELP dnsdist_latency_avg10000 Average response latency in microseconds␣
↪of the last 10000 packets
```

(continues on next page)

(continued from previous page)

```
# TYPE dnsdist_latency_avg10000 gauge
dnsdist_latency_avg10000 0
# HELP dnsdist_latency_avg1000000 Average response latency in microseconds␣
↪of the last 1000000 packets
# TYPE dnsdist_latency_avg1000000 gauge
dnsdist_latency_avg1000000 0
# HELP dnsdist_latency_tcp_avg100 Average response latency, in␣
↪microseconds, of the last 100 packets received over TCP
# TYPE dnsdist_latency_tcp_avg100 gauge
dnsdist_latency_tcp_avg100 0
# HELP dnsdist_latency_tcp_avg1000 Average response latency, in␣
↪microseconds, of the last 1000 packets received over TCP
# TYPE dnsdist_latency_tcp_avg1000 gauge
dnsdist_latency_tcp_avg1000 0
# HELP dnsdist_latency_tcp_avg10000 Average response latency, in␣
↪microseconds, of the last 10000 packets received over TCP
# TYPE dnsdist_latency_tcp_avg10000 gauge
dnsdist_latency_tcp_avg10000 0
# HELP dnsdist_latency_tcp_avg1000000 Average response latency, in␣
↪microseconds, of the last 1000000 packets received over TCP
# TYPE dnsdist_latency_tcp_avg1000000 gauge
dnsdist_latency_tcp_avg1000000 0
# HELP dnsdist_latency_dot_avg100 Average response latency, in␣
↪microseconds, of the last 100 packets received over DoT
# TYPE dnsdist_latency_dot_avg100 gauge
dnsdist_latency_dot_avg100 0
# HELP dnsdist_latency_dot_avg1000 Average response latency, in␣
↪microseconds, of the last 1000 packets received over DoT
# TYPE dnsdist_latency_dot_avg1000 gauge
dnsdist_latency_dot_avg1000 0
# HELP dnsdist_latency_dot_avg10000 Average response latency, in␣
↪microseconds, of the last 10000 packets received over DoT
# TYPE dnsdist_latency_dot_avg10000 gauge
dnsdist_latency_dot_avg10000 0
# HELP dnsdist_latency_dot_avg1000000 Average response latency, in␣
↪microseconds, of the last 1000000 packets received over DoT
# TYPE dnsdist_latency_dot_avg1000000 gauge
dnsdist_latency_dot_avg1000000 0
# HELP dnsdist_latency_doh_avg100 Average response latency, in␣
↪microseconds, of the last 100 packets received over DoH
# TYPE dnsdist_latency_doh_avg100 gauge
dnsdist_latency_doh_avg100 0
# HELP dnsdist_latency_doh_avg1000 Average response latency, in␣
↪microseconds, of the last 1000 packets received over DoH
# TYPE dnsdist_latency_doh_avg1000 gauge
dnsdist_latency_doh_avg1000 0
# HELP dnsdist_latency_doh_avg10000 Average response latency, in␣
↪microseconds, of the last 10000 packets received over DoH
# TYPE dnsdist_latency_doh_avg10000 gauge
dnsdist_latency_doh_avg10000 0
# HELP dnsdist_latency_doh_avg1000000 Average response latency, in␣
↪microseconds, of the last 1000000 packets received over DoH
# TYPE dnsdist_latency_doh_avg1000000 gauge
dnsdist_latency_doh_avg1000000 0
# HELP dnsdist_uptime Uptime of the dnsdist process in seconds
# TYPE dnsdist_uptime gauge
dnsdist_uptime 19
# HELP dnsdist_real_memory_usage Current memory usage in bytes
# TYPE dnsdist_real_memory_usage gauge
dnsdist_real_memory_usage 52269056
# HELP dnsdist_udp_in_errors From /proc/net/snmp InErrors
```

(continues on next page)

(continued from previous page)

```
# TYPE dnsdist_udp_in_errors counter
dnsdist_udp_in_errors 0
# HELP dnsdist_udp_noport_errors From /proc/net/snmp NoPorts
# TYPE dnsdist_udp_noport_errors counter
dnsdist_udp_noport_errors 86
# HELP dnsdist_udp_recvbuf_errors From /proc/net/snmp RcvbufErrors
# TYPE dnsdist_udp_recvbuf_errors counter
dnsdist_udp_recvbuf_errors 0
# HELP dnsdist_udp_sndbuf_errors From /proc/net/snmp SndbufErrors
# TYPE dnsdist_udp_sndbuf_errors counter
dnsdist_udp_sndbuf_errors 0
# HELP dnsdist_udp_in_csum_errors From /proc/net/snmp InCsumErrors
# TYPE dnsdist_udp_in_csum_errors counter
dnsdist_udp_in_csum_errors 0
# HELP dnsdist_udp6_in_errors From /proc/net/snmp6 Udp6InErrors
# TYPE dnsdist_udp6_in_errors counter
dnsdist_udp6_in_errors 0
# HELP dnsdist_udp6_recvbuf_errors From /proc/net/snmp6 Udp6RcvbufErrors
# TYPE dnsdist_udp6_recvbuf_errors counter
dnsdist_udp6_recvbuf_errors 0
# HELP dnsdist_udp6_sndbuf_errors From /proc/net/snmp6 Udp6SndbufErrors
# TYPE dnsdist_udp6_sndbuf_errors counter
dnsdist_udp6_sndbuf_errors 0
# HELP dnsdist_udp6_noport_errors From /proc/net/snmp6 Udp6NoPorts
# TYPE dnsdist_udp6_noport_errors counter
dnsdist_udp6_noport_errors 195
# HELP dnsdist_udp6_in_csum_errors From /proc/net/snmp6 Udp6InCsumErrors
# TYPE dnsdist_udp6_in_csum_errors counter
dnsdist_udp6_in_csum_errors 0
# HELP dnsdist_tcp_listen_overflows From /proc/net/netstat ListenOverflows
# TYPE dnsdist_tcp_listen_overflows counter
dnsdist_tcp_listen_overflows 0
# HELP dnsdist_noncompliant_queries Number of queries dropped as non-
↪compliant
# TYPE dnsdist_noncompliant_queries counter
dnsdist_noncompliant_queries 0
# HELP dnsdist_noncompliant_responses Number of answers from a backend␣
↪dropped as non-compliant
# TYPE dnsdist_noncompliant_responses counter
dnsdist_noncompliant_responses 0
# HELP dnsdist_proxy_protocol_invalid Number of queries dropped because of␣
↪an invalid Proxy Protocol header
# TYPE dnsdist_proxy_protocol_invalid counter
dnsdist_proxy_protocol_invalid 0
# HELP dnsdist_rdqueries Number of received queries with the recursion␣
↪desired bit set
# TYPE dnsdist_rdqueries counter
dnsdist_rdqueries 0
# HELP dnsdist_empty_queries Number of empty queries received from clients
# TYPE dnsdist_empty_queries counter
dnsdist_empty_queries 0
# HELP dnsdist_cache_hits Number of times an answer was retrieved from␣
↪cache
# TYPE dnsdist_cache_hits counter
dnsdist_cache_hits 0
# HELP dnsdist_cache_misses Number of times an answer not found in the␣
↪cache
# TYPE dnsdist_cache_misses counter
dnsdist_cache_misses 0
# HELP dnsdist_cpu_iowait Time waiting for I/O to complete by the whole␣
↪system, in units of USER_HZ
```

(continues on next page)

(continued from previous page)

```
# TYPE dnsdist_cpu_iowait counter
dnsdist_cpu_iowait 0
# HELP dnsdist_cpu_steal Stolen time, which is the time spent by the whole␣
↪system in other operating systems when running in a virtualized␣
↪environment, in units of USER_HZ
# TYPE dnsdist_cpu_steal counter
dnsdist_cpu_steal 0
# HELP dnsdist_cpu_sys_msec Milliseconds spent by dnsdist in the system␣
↪state
# TYPE dnsdist_cpu_sys_msec counter
dnsdist_cpu_sys_msec 38
# HELP dnsdist_cpu_user_msec Milliseconds spent by dnsdist in the user␣
↪state
# TYPE dnsdist_cpu_user_msec counter
dnsdist_cpu_user_msec 38
# HELP dnsdist_fd_usage Number of currently used file descriptors
# TYPE dnsdist_fd_usage gauge
dnsdist_fd_usage 32
# HELP dnsdist_dyn_blocked Number of queries dropped because of a dynamic␣
↪block
# TYPE dnsdist_dyn_blocked counter
dnsdist_dyn_blocked 0
# HELP dnsdist_dyn_block_nmg_size Number of dynamic blocks entries
# TYPE dnsdist_dyn_block_nmg_size gauge
dnsdist_dyn_block_nmg_size 0
# HELP dnsdist_security_status Security status of this software. 0=unknown,
↪ 1=OK, 2=upgrade recommended, 3=upgrade mandatory
# TYPE dnsdist_security_status gauge
dnsdist_security_status 0
# HELP dnsdist_doh_query_pipe_full Number of DoH queries dropped because␣
↪the internal pipe used to distribute queries was full
# TYPE dnsdist_doh_query_pipe_full counter
dnsdist_doh_query_pipe_full 0
# HELP dnsdist_doh_response_pipe_full Number of DoH responses dropped␣
↪because the internal pipe used to distribute responses was full
# TYPE dnsdist_doh_response_pipe_full counter
dnsdist_doh_response_pipe_full 0
# HELP dnsdist_outgoing_doh_query_pipe_full Number of outgoing DoH queries␣
↪dropped because the internal pipe used to distribute queries was full
# TYPE dnsdist_outgoing_doh_query_pipe_full counter
dnsdist_outgoing_doh_query_pipe_full 0
# HELP dnsdist_tcp_query_pipe_full Number of TCP queries dropped because␣
↪the internal pipe used to distribute queries was full
# TYPE dnsdist_tcp_query_pipe_full counter
dnsdist_tcp_query_pipe_full 0
# HELP dnsdist_tcp_cross_protocol_query_pipe_full Number of TCP cross-
↪protocol queries dropped because the internal pipe used to distribute␣
↪queries was full
# TYPE dnsdist_tcp_cross_protocol_query_pipe_full counter
dnsdist_tcp_cross_protocol_query_pipe_full 0
# HELP dnsdist_tcp_cross_protocol_response_pipe_full Number of TCP cross-
↪protocol responses dropped because the internal pipe used to distribute␣
↪queries was full
# TYPE dnsdist_tcp_cross_protocol_response_pipe_full counter
dnsdist_tcp_cross_protocol_response_pipe_full 0
# HELP dnsdist_latency Histogram of responses by latency (in milliseconds)
# TYPE dnsdist_latency histogram
dnsdist_latency_bucket{le="1"} 0
dnsdist_latency_bucket{le="10"} 0
dnsdist_latency_bucket{le="50"} 0
dnsdist_latency_bucket{le="100"} 0
```

(continues on next page)

(continued from previous page)

```
dnsdist_latency_bucket{le="1000"} 0
dnsdist_latency_bucket{le="+Inf"} 0
dnsdist_latency_sum 0
dnsdist_latency_count 0
# HELP dnsdist_server_status Whether this backend is up (1) or down (0)
# TYPE dnsdist_server_status gauge
# HELP dnsdist_server_queries Amount of queries relayed to server
# TYPE dnsdist_server_queries counter
# HELP dnsdist_server_responses Amount of responses received from this␣
↪server
# TYPE dnsdist_server_responses counter
# HELP dnsdist_server_noncompliantresponses Amount of non-compliant␣
↪responses received from this server
# TYPE dnsdist_server_noncompliantresponses counter
# HELP dnsdist_server_drops Amount of queries not answered by server
# TYPE dnsdist_server_drops counter
# HELP dnsdist_server_latency Server's latency when answering questions in␣
↪milliseconds
# TYPE dnsdist_server_latency gauge
# HELP dnsdist_server_senderrors Total number of OS send errors while␣
↪relaying queries
# TYPE dnsdist_server_senderrors counter
# HELP dnsdist_server_outstanding Current number of queries that are␣
↪waiting for a backend response
# TYPE dnsdist_server_outstanding gauge
# HELP dnsdist_server_order The order in which this server is picked
# TYPE dnsdist_server_order gauge
# HELP dnsdist_server_weight The weight within the order in which this␣
↪server is picked
# TYPE dnsdist_server_weight gauge
# HELP dnsdist_server_tcpdiedsendingquery The number of TCP I/O errors␣
↪while sending the query
# TYPE dnsdist_server_tcpdiedsendingquery counter
# HELP dnsdist_server_tcpdiedreadingresponse The number of TCP I/O errors␣
↪while reading the response
# TYPE dnsdist_server_tcpdiedreadingresponse counter
# HELP dnsdist_server_tcpgaveup The number of TCP connections failing␣
↪after too many attempts
# TYPE dnsdist_server_tcpgaveup counter
# HELP dnsdist_server_tcpconnecttimeouts The number of TCP connect timeouts
# TYPE dnsdist_server_tcpconnecttimeouts counter
# HELP dnsdist_server_tcpreadtimeouts The number of TCP read timeouts
# TYPE dnsdist_server_tcpreadtimeouts counter
# HELP dnsdist_server_tcpwritetimeouts The number of TCP write timeouts
# TYPE dnsdist_server_tcpwritetimeouts counter
# HELP dnsdist_server_tcpcurrentconnections The number of current TCP␣
↪connections
# TYPE dnsdist_server_tcpcurrentconnections gauge
# HELP dnsdist_server_tcpmaxconcurrentconnections The maximum number of␣
↪concurrent TCP connections
# TYPE dnsdist_server_tcpmaxconcurrentconnections counter
# HELP dnsdist_server_tcptoomanyconcurrentconnections Number of times we␣
↪had to enforce the maximum number of concurrent TCP connections
# TYPE dnsdist_server_tcptoomanyconcurrentconnections counter
# HELP dnsdist_server_tcpnewconnections The number of established TCP␣
↪connections in total
# TYPE dnsdist_server_tcpnewconnections counter
# HELP dnsdist_server_tcpreusedconnections The number of times a TCP␣
↪connection has been reused
# TYPE dnsdist_server_tcpreusedconnections counter
# HELP dnsdist_server_tcpavgqueriesperconn The average number of queries␣
↪per TCP connection
```

(continues on next page)

(continued from previous page)

```
# TYPE dnsdist_server_tcpavgqueriesperconn gauge
# HELP dnsdist_server_tcpavgconnduration The average duration of a TCP␣
↪connection (ms)
# TYPE dnsdist_server_tcpavgconnduration gauge
# HELP dnsdist_server_tlsresumptions The number of times a TLS session has␣
↪been resumed
# TYPE dnsdist_server_tlsresumptions counter
# HELP dnsdist_server_tcplatency Server's latency when answering TCP␣
↪questions in milliseconds
# TYPE dnsdist_server_tcplatency gauge
dnsdist_server_status{server="9_9_9_9:443",address="9.9.9.9:443"} 1
dnsdist_server_queries{server="9_9_9_9:443",address="9.9.9.9:443"} 0
dnsdist_server_responses{server="9_9_9_9:443",address="9.9.9.9:443"} 0
dnsdist_server_noncompliantresponses{server="9_9_9_9:443",address="9.9.9.
↪9:443"} 0
dnsdist_server_drops{server="9_9_9_9:443",address="9.9.9.9:443"} 0
dnsdist_server_latency{server="9_9_9_9:443",address="9.9.9.9:443"} 0
dnsdist_server_tcplatency{server="9_9_9_9:443",address="9.9.9.9:443"} 0
dnsdist_server_senderrors{server="9_9_9_9:443",address="9.9.9.9:443"} 0
dnsdist_server_outstanding{server="9_9_9_9:443",address="9.9.9.9:443"} 0
dnsdist_server_order{server="9_9_9_9:443",address="9.9.9.9:443"} 1
dnsdist_server_weight{server="9_9_9_9:443",address="9.9.9.9:443"} 1
dnsdist_server_tcpdiedsendingquery{server="9_9_9_9:443",address="9.9.9.
↪9:443"} 0
dnsdist_server_tcpdiedreadingresponse{server="9_9_9_9:443",address="9.9.9.
↪9:443"} 0
dnsdist_server_tcpgaveup{server="9_9_9_9:443",address="9.9.9.9:443"} 0
dnsdist_server_tcpreadtimeouts{server="9_9_9_9:443",address="9.9.9.9:443"}␣
↪0
dnsdist_server_tcpwritetimeouts{server="9_9_9_9:443",address="9.9.9.9:443"}
↪ 0
dnsdist_server_tcpconnecttimeouts{server="9_9_9_9:443",address="9.9.9.9:443
↪"} 0
dnsdist_server_tcpcurrentconnections{server="9_9_9_9:443",address="9.9.9.
↪9:443"} 0
dnsdist_server_tcpmaxconcurrentconnections{server="9_9_9_9:443",address="9.
↪9.9.9:443"} 1
dnsdist_server_tcptoomanyconcurrentconnections{server="9_9_9_9:443",
↪address="9.9.9.9:443"} 0
dnsdist_server_tcpnewconnections{server="9_9_9_9:443",address="9.9.9.9:443
↪"} 19
dnsdist_server_tcpreusedconnections{server="9_9_9_9:443",address="9.9.9.
↪9:443"} 0
dnsdist_server_tcpavgqueriesperconn{server="9_9_9_9:443",address="9.9.9.
↪9:443"} 0.173831
dnsdist_server_tcpavgconnduration{server="9_9_9_9:443",address="9.9.9.9:443
↪"} 3.92628
dnsdist_server_tlsresumptions{server="9_9_9_9:443",address="9.9.9.9:443"}␣
↪18
# HELP dnsdist_frontend_queries Amount of queries received by this frontend
# TYPE dnsdist_frontend_queries counter
# HELP dnsdist_frontend_noncompliantqueries Amount of non-compliant␣
↪queries received by this frontend
# TYPE dnsdist_frontend_noncompliantqueries counter
# HELP dnsdist_frontend_responses Amount of responses sent by this frontend
# TYPE dnsdist_frontend_responses counter
# HELP dnsdist_frontend_tcpdiedreadingquery Amount of TCP connections␣
↪terminated while reading the query from the client
# TYPE dnsdist_frontend_tcpdiedreadingquery counter
# HELP dnsdist_frontend_tcpdiedsendingresponse Amount of TCP connections␣
↪terminated while sending a response to the client
```

(continues on next page)

(continued from previous page)

```
# TYPE dnsdist_frontend_tcpdiedsendingresponse counter
# HELP dnsdist_frontend_tcpgaveup Amount of TCP connections terminated␣
↪after too many attempts to get a connection to the backend
# TYPE dnsdist_frontend_tcpgaveup counter
# HELP dnsdist_frontend_tcpclienttimeouts Amount of TCP connections␣
↪terminated by a timeout while reading from the client
# TYPE dnsdist_frontend_tcpclienttimeouts counter
# HELP dnsdist_frontend_tcpdownstreamtimeouts Amount of TCP connections␣
↪terminated by a timeout while reading from the backend
# TYPE dnsdist_frontend_tcpdownstreamtimeouts counter
# HELP dnsdist_frontend_tcpcurrentconnections Amount of current incoming␣
↪TCP connections from clients
# TYPE dnsdist_frontend_tcpcurrentconnections gauge
# HELP dnsdist_frontend_tcpmaxconcurrentconnections Maximum number of␣
↪concurrent incoming TCP connections from clients
# TYPE dnsdist_frontend_tcpmaxconcurrentconnections counter
# HELP dnsdist_frontend_tcpavgqueriesperconnection The average number of␣
↪queries per TCP connection
# TYPE dnsdist_frontend_tcpavgqueriesperconnection gauge
# HELP dnsdist_frontend_tcpavgconnectionduration The average duration of a␣
↪TCP connection (ms)
# TYPE dnsdist_frontend_tcpavgconnectionduration gauge
# HELP dnsdist_frontend_tlsqueries Number of queries received by dnsdist␣
↪over TLS, by TLS version
# TYPE dnsdist_frontend_tlsqueries counter
# HELP dnsdist_frontend_tlsnewsessions Amount of new TLS sessions␣
↪negotiated
# TYPE dnsdist_frontend_tlsnewsessions counter
# HELP dnsdist_frontend_tlsresumptions Amount of TLS sessions resumed
# TYPE dnsdist_frontend_tlsresumptions counter
# HELP dnsdist_frontend_tlsunknownticketkeys Amount of attempts to resume␣
↪TLS session from an unknown key (possibly expired)
# TYPE dnsdist_frontend_tlsunknownticketkeys counter
# HELP dnsdist_frontend_tlsinactiveticketkeys Amount of TLS sessions␣
↪resumed from an inactive key
# TYPE dnsdist_frontend_tlsinactiveticketkeys counter
# HELP dnsdist_frontend_tlshandshakefailures Amount of TLS handshake␣
↪failures
# TYPE dnsdist_frontend_tlshandshakefailures counter
dnsdist_frontend_queries{frontend="127.0.0.1:853",proto="TCP (DNS over TLS)
↪",thread="0"} 0
dnsdist_frontend_noncompliantqueries{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0"} 0
dnsdist_frontend_responses{frontend="127.0.0.1:853",proto="TCP (DNS over␣
↪TLS)",thread="0"} 0
dnsdist_frontend_tcpdiedreadingquery{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0"} 0
dnsdist_frontend_tcpdiedsendingresponse{frontend="127.0.0.1:853",proto=
↪"TCP (DNS over TLS)",thread="0"} 0
dnsdist_frontend_tcpgaveup{frontend="127.0.0.1:853",proto="TCP (DNS over␣
↪TLS)",thread="0"} 0
dnsdist_frontend_tcpclienttimeouts{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0"} 0
dnsdist_frontend_tcpdownstreamtimeouts{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0"} 0
dnsdist_frontend_tcpcurrentconnections{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0"} 0
dnsdist_frontend_tcpmaxconcurrentconnections{frontend="127.0.0.1:853",
↪proto="TCP (DNS over TLS)",thread="0"} 0
dnsdist_frontend_tcpavgqueriesperconnection{frontend="127.0.0.1:853",proto=
↪"TCP (DNS over TLS)",thread="0"} 0
```

(continues on next page)

(continued from previous page)

```
dnsdist_frontend_tcpavgconnectionduration{frontend="127.0.0.1:853",proto=
↪"TCP (DNS over TLS)",thread="0"} 0
dnsdist_frontend_tlsnewsessions{frontend="127.0.0.1:853",proto="TCP (DNS␣
↪over TLS)",thread="0"} 0
dnsdist_frontend_tlsresumptions{frontend="127.0.0.1:853",proto="TCP (DNS␣
↪over TLS)",thread="0"} 0
dnsdist_frontend_tlsunknownticketkeys{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0"} 0
dnsdist_frontend_tlsinactiveticketkeys{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0"} 0
dnsdist_frontend_tlsqueries{frontend="127.0.0.1:853",proto="TCP (DNS over␣
↪TLS)",thread="0",tls="tls10"} 0
dnsdist_frontend_tlsqueries{frontend="127.0.0.1:853",proto="TCP (DNS over␣
↪TLS)",thread="0",tls="tls11"} 0
dnsdist_frontend_tlsqueries{frontend="127.0.0.1:853",proto="TCP (DNS over␣
↪TLS)",thread="0",tls="tls12"} 0
dnsdist_frontend_tlsqueries{frontend="127.0.0.1:853",proto="TCP (DNS over␣
↪TLS)",thread="0",tls="tls13"} 0
dnsdist_frontend_tlsqueries{frontend="127.0.0.1:853",proto="TCP (DNS over␣
↪TLS)",thread="0",tls="unknown"} 0
dnsdist_frontend_tlshandshakefailures{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0",error="dhKeyTooSmall"} 0
dnsdist_frontend_tlshandshakefailures{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0",error="inappropriateFallBack"} 0
dnsdist_frontend_tlshandshakefailures{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0",error="noSharedCipher"} 0
dnsdist_frontend_tlshandshakefailures{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0",error="unknownCipherType"} 0
dnsdist_frontend_tlshandshakefailures{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0",error="unknownKeyExchangeType"} 0
dnsdist_frontend_tlshandshakefailures{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0",error="unknownProtocol"} 0
dnsdist_frontend_tlshandshakefailures{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0",error="unsupportedEC"} 0
dnsdist_frontend_tlshandshakefailures{frontend="127.0.0.1:853",proto="TCP␣
↪(DNS over TLS)",thread="0",error="unsupportedProtocol"} 0
dnsdist_frontend_queries{frontend="[::1]:443",proto="TCP (DNS over HTTPS)",
↪thread="0"} 0
dnsdist_frontend_noncompliantqueries{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0"} 0
dnsdist_frontend_responses{frontend="[::1]:443",proto="TCP (DNS over HTTPS)
↪",thread="0"} 0
dnsdist_frontend_tcpdiedreadingquery{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0"} 0
dnsdist_frontend_tcpdiedsendingresponse{frontend="[::1]:443",proto="TCP␣
↪(DNS over HTTPS)",thread="0"} 0
dnsdist_frontend_tcpgaveup{frontend="[::1]:443",proto="TCP (DNS over HTTPS)
↪",thread="0"} 0
dnsdist_frontend_tcpclienttimeouts{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0"} 0
dnsdist_frontend_tcpdownstreamtimeouts{frontend="[::1]:443",proto="TCP␣
↪(DNS over HTTPS)",thread="0"} 0
dnsdist_frontend_tcpcurrentconnections{frontend="[::1]:443",proto="TCP␣
↪(DNS over HTTPS)",thread="0"} 0
dnsdist_frontend_tcpmaxconcurrentconnections{frontend="[::1]:443",proto=
↪"TCP (DNS over HTTPS)",thread="0"} 0
dnsdist_frontend_tcpavgqueriesperconnection{frontend="[::1]:443",proto=
↪"TCP (DNS over HTTPS)",thread="0"} 0
dnsdist_frontend_tcpavgconnectionduration{frontend="[::1]:443",proto="TCP␣
↪(DNS over HTTPS)",thread="0"} 0
dnsdist_frontend_tlsnewsessions{frontend="[::1]:443",proto="TCP (DNS over␣
↪HTTPS)",thread="0"} 0
```

(continues on next page)

(continued from previous page)

```
dnsdist_frontend_tlsresumptions{frontend="[::1]:443",proto="TCP (DNS over␣
↪HTTPS)",thread="0"} 0
dnsdist_frontend_tlsunknownticketkeys{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0"} 0
dnsdist_frontend_tlsinactiveticketkeys{frontend="[::1]:443",proto="TCP␣
↪(DNS over HTTPS)",thread="0"} 0
dnsdist_frontend_tlsqueries{frontend="[::1]:443",proto="TCP (DNS over␣
↪HTTPS)",thread="0",tls="tls10"} 0
dnsdist_frontend_tlsqueries{frontend="[::1]:443",proto="TCP (DNS over␣
↪HTTPS)",thread="0",tls="tls11"} 0
dnsdist_frontend_tlsqueries{frontend="[::1]:443",proto="TCP (DNS over␣
↪HTTPS)",thread="0",tls="tls12"} 0
dnsdist_frontend_tlsqueries{frontend="[::1]:443",proto="TCP (DNS over␣
↪HTTPS)",thread="0",tls="tls13"} 0
dnsdist_frontend_tlsqueries{frontend="[::1]:443",proto="TCP (DNS over␣
↪HTTPS)",thread="0",tls="unknown"} 0
dnsdist_frontend_tlshandshakefailures{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0",error="dhKeyTooSmall"} 0
dnsdist_frontend_tlshandshakefailures{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0",error="inappropriateFallBack"} 0
dnsdist_frontend_tlshandshakefailures{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0",error="noSharedCipher"} 0
dnsdist_frontend_tlshandshakefailures{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0",error="unknownCipherType"} 0
dnsdist_frontend_tlshandshakefailures{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0",error="unknownKeyExchangeType"} 0
dnsdist_frontend_tlshandshakefailures{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0",error="unknownProtocol"} 0
dnsdist_frontend_tlshandshakefailures{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0",error="unsupportedEC"} 0
dnsdist_frontend_tlshandshakefailures{frontend="[::1]:443",proto="TCP (DNS␣
↪over HTTPS)",thread="0",error="unsupportedProtocol"} 0
dnsdist_frontend_queries{frontend="127.0.0.1:53",proto="UDP",thread="0"} 0
dnsdist_frontend_noncompliantqueries{frontend="127.0.0.1:53",proto="UDP",
↪thread="0"} 0
dnsdist_frontend_responses{frontend="127.0.0.1:53",proto="UDP",thread="0"}␣
↪0
dnsdist_frontend_queries{frontend="127.0.0.1:53",proto="TCP",thread="0"} 0
dnsdist_frontend_noncompliantqueries{frontend="127.0.0.1:53",proto="TCP",
↪thread="0"} 0
dnsdist_frontend_responses{frontend="127.0.0.1:53",proto="TCP",thread="0"}␣
↪0
dnsdist_frontend_tcpdiedreadingquery{frontend="127.0.0.1:53",proto="TCP",
↪thread="0"} 0
dnsdist_frontend_tcpdiedsendingresponse{frontend="127.0.0.1:53",proto="TCP
↪",thread="0"} 0
dnsdist_frontend_tcpgaveup{frontend="127.0.0.1:53",proto="TCP",thread="0"}␣
↪0
dnsdist_frontend_tcpclienttimeouts{frontend="127.0.0.1:53",proto="TCP",
↪thread="0"} 0
dnsdist_frontend_tcpdownstreamtimeouts{frontend="127.0.0.1:53",proto="TCP",
↪thread="0"} 0
dnsdist_frontend_tcpcurrentconnections{frontend="127.0.0.1:53",proto="TCP",
↪thread="0"} 0
dnsdist_frontend_tcpmaxconcurrentconnections{frontend="127.0.0.1:53",proto=
↪"TCP",thread="0"} 0
dnsdist_frontend_tcpavgqueriesperconnection{frontend="127.0.0.1:53",proto=
↪"TCP",thread="0"} 0
dnsdist_frontend_tcpavgconnectionduration{frontend="127.0.0.1:53",proto=
↪"TCP",thread="0"} 0
# HELP dnsdist_frontend_http_connects Number of DoH TCP connections␣
↪established to this frontend
```

(continues on next page)

(continued from previous page)

```
# TYPE dnsdist_frontend_http_connects counter
# HELP dnsdist_frontend_doh_http_method_queries Number of DoH queries␣
↪received by dnsdist, by HTTP method
# TYPE dnsdist_frontend_doh_http_method_queries counter
# HELP dnsdist_frontend_doh_http_version_queries Number of DoH queries␣
↪received by dnsdist, by HTTP version
# TYPE dnsdist_frontend_doh_http_version_queries counter
# HELP dnsdist_frontend_doh_bad_requests Number of requests that could not␣
↪be converted to a DNS query
# TYPE dnsdist_frontend_doh_bad_requests counter
# HELP dnsdist_frontend_doh_responses Number of responses sent, by type
# TYPE dnsdist_frontend_doh_responses counter
# HELP dnsdist_frontend_doh_version_status_responses Number of requests␣
↪that could not be converted to a DNS query
# TYPE dnsdist_frontend_doh_version_status_responses counter
dnsdist_frontend_http_connects{frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_http_method_queries{method="get",frontend="[::1]:443",
↪thread="0"} 0
dnsdist_frontend_doh_http_method_queries{method="post",frontend="[::1]:443
↪",thread="0"} 0
dnsdist_frontend_doh_http_version_queries{version="1",frontend="[::1]:443",
↪thread="0"} 0
dnsdist_frontend_doh_http_version_queries{version="2",frontend="[::1]:443",
↪thread="0"} 0
dnsdist_frontend_doh_bad_requests{frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_responses{type="error",frontend="[::1]:443",thread="0
↪"} 0
dnsdist_frontend_doh_responses{type="redirect",frontend="[::1]:443",thread=
↪"0"} 0
dnsdist_frontend_doh_responses{type="valid",frontend="[::1]:443",thread="0
↪"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="1",status="200",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="1",status="400",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="1",status="403",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="1",status="500",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="1",status="502",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="1",status="other
↪",frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="2",status="200",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="2",status="400",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="2",status="403",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="2",status="500",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="2",status="502",
↪frontend="[::1]:443",thread="0"} 0
dnsdist_frontend_doh_version_status_responses{httpversion="2",status="other
↪",frontend="[::1]:443",thread="0"} 0
# HELP dnsdist_pool_servers Number of servers in that pool
# TYPE dnsdist_pool_servers gauge
# HELP dnsdist_pool_active_servers Number of available servers in that pool
# TYPE dnsdist_pool_active_servers gauge
# HELP dnsdist_pool_cache_size Maximum number of entries that this cache␣
↪can hold
```

(continues on next page)

(continued from previous page)

```
# TYPE dnsdist_pool_cache_size gauge
# HELP dnsdist_pool_cache_entries Number of entries currently present in␣
↪that cache
# TYPE dnsdist_pool_cache_entries gauge
# HELP dnsdist_pool_cache_hits Number of hits from that cache
# TYPE dnsdist_pool_cache_hits counter
# HELP dnsdist_pool_cache_misses Number of misses from that cache
# TYPE dnsdist_pool_cache_misses counter
# HELP dnsdist_pool_cache_deferred_inserts Number of insertions into that␣
↪cache skipped because it was already locked
# TYPE dnsdist_pool_cache_deferred_inserts counter
# HELP dnsdist_pool_cache_deferred_lookups Number of lookups into that␣
↪cache skipped because it was already locked
# TYPE dnsdist_pool_cache_deferred_lookups counter
# HELP dnsdist_pool_cache_lookup_collisions Number of lookups into that␣
↪cache that triggered a collision (same hash but different entry)
# TYPE dnsdist_pool_cache_lookup_collisions counter
# HELP dnsdist_pool_cache_insert_collisions Number of insertions into that␣
↪cache that triggered a collision (same hash but different entry)
# TYPE dnsdist_pool_cache_insert_collisions counter
# HELP dnsdist_pool_cache_ttl_too_shorts Number of insertions into that␣
↪cache skipped because the TTL of the answer was not long enough
# TYPE dnsdist_pool_cache_ttl_too_shorts counter
# HELP dnsdist_pool_cache_cleanup_count_total Number of times the cache␣
↪has been scanned to remove expired entries, if any
# TYPE dnsdist_pool_cache_cleanup_count_total counter
dnsdist_pool_servers{pool="_default_"} 1
dnsdist_pool_active_servers{pool="_default_"} 1
dnsdist_pool_cache_size{pool="_default_"} 100
dnsdist_pool_cache_entries{pool="_default_"} 0
dnsdist_pool_cache_hits{pool="_default_"} 0
dnsdist_pool_cache_misses{pool="_default_"} 0
dnsdist_pool_cache_deferred_inserts{pool="_default_"} 0
dnsdist_pool_cache_deferred_lookups{pool="_default_"} 0
dnsdist_pool_cache_lookup_collisions{pool="_default_"} 0
dnsdist_pool_cache_insert_collisions{pool="_default_"} 0
dnsdist_pool_cache_ttl_too_shorts{pool="_default_"} 0
dnsdist_pool_cache_cleanup_count_total{pool="_default_"} 0
# HELP dnsdist_rule_hits Number of hits of that rule
# TYPE dnsdist_rule_hits counter
# HELP dnsdist_dynblocks_nmg_top_offenders_hits_per_second Number of hits␣
↪per second blocked by Dynamic Blocks (netmasks) for the top offenders,␣
↪averaged over the last 60s
# TYPE dnsdist_dynblocks_nmg_top_offenders_hits_per_second gauge
# HELP dnsdist_dynblocks_smt_top_offenders_hits_per_second Number of this␣
↪per second blocked by Dynamic Blocks (suffixes) for the top offenders,␣
↪averaged over the last 60s
# TYPE dnsdist_dynblocks_smt_top_offenders_hits_per_second gauge
# HELP dnsdist_info Info from dnsdist, value is always 1
# TYPE dnsdist_info gauge
dnsdist_info{version="1.7.3"} 1
```

Example prometheus configuration:

This is just the scrape job description, for details see the prometheus documentation.

```
job_name: dnsdist
scrape_interval: 10s
scrape_timeout: 2s
metrics_path: /metrics
basic_auth:
```

(continues on next page)

(continued from previous page)

```
  username: dontcare
  password: yoursecret
```

DELETE /api/v1/cache?pool=<pool-name>&name=<dns-name>[&type=<dns-type>][&suffix=]

New in version 1.8.0.

Allows removing entries from a cache. The pool to which the cache is associated should be specified in the `pool` parameter, and the name to remove in the `name` parameter. By default only entries matching the exact name will be removed, but it is possible to remove all entries below that name by passing the `suffix` parameter set to any value. By default entries for all types for the name are removed, but it is possible to only remove entries for a specific type by passing the `type` parameter set to the requested type. Supported values are DNS type names as a strings (`AAAA`), or numerical values (as either `#64` or `TYPE64`).

Example request:

```
DELETE /api/v1/cache?pool=&name=free.fr HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 0
Host: localhost:8080
X-API-Key: supersecretAPIkey
```

Example response:

```
HTTP/1.1 200 OK
Connection: close
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-
↪inline'
Content-Type: application/json
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 1; mode=block

{
    "count": "1",
    "status": "purged"
}
```

GET /api/v1/servers/localhost

Get a quick overview of several parameters.

Response JSON Object

- **acl** (*string*) – A string of comma-separated netmasks currently allowed by the *ACL*.
- **cache-hit-response-rules** (*list*) – A list of *ResponseRule* objects applied on cache hits
- **self-answered-response-rules** (*list*) – A list of *ResponseRule* objects applied on self-answered queries
- **daemon_type** (*string*) – The type of daemon, always “dnsdist”
- **frontends** (*list*) – A list of *Frontend* objects
- **pools** (*list*) – A list of *Pool* objects
- **response-rules** (*list*) – A list of *ResponseRule* objects
- **rules** (*list*) – A list of *Rule* objects

- **servers** (*list*) – A list of *Server* objects
- **version** (*string*) – The running version of dnsdist

GET /api/v1/servers/localhost/statistics

Returns a list of all statistics as *StatisticItem*.

GET /api/v1/servers/localhost/config

Returns a list of *ConfigSetting* objects.

GET /api/v1/servers/localhost/config/allow-from

Gets you the `allow-from` *ConfigSetting*, who's value is a list of strings of all the netmasks in the *ACL*.

Example request:

```
GET /api/v1/servers/localhost/config/allow-from HTTP/1.1
X-API-Key: supersecretAPIkey
```

Example response:

```
HTTP/1.1 200 OK
Connection: close
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-
↪inline'
Content-Type: application/json
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 1; mode=block

{
    "name": "allow-from",
    "type": "ConfigSetting",
    "value": [
        "fc00::/7",
        "169.254.0.0/16",
        "100.64.0.0/10",
        "fe80::/10",
        "10.0.0.0/8",
        "127.0.0.0/8",
        "::1/128",
        "172.16.0.0/12",
        "192.168.0.0/16"
    ]
}
```

PUT /api/v1/servers/localhost/config/allow-from

Allows you to update the `allow-from` *ACL* with a list of netmasks.

Make sure you made the API writable using *setAPIWritable()*. Changes to the ACL are directly applied, no restart is required.

Example request:

```
PUT /api/v1/servers/localhost/config/allow-from HTTP/1.1
Content-Length: 37
Content-Type: application/json
X-API-Key: supersecretAPIkey

{
    "value": [
```

(continues on next page)

(continued from previous page)

```
        "127.0.0.0/8",
        "::1/128"
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Connection: close
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-
↪inline'
Content-Type: application/json
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: deny
X-Permitted-Cross-Domain-Policies: none
X-Xss-Protection: 1; mode=block

{
    "name": "allow-from",
    "type": "ConfigSetting",
    "value": [
        "127.0.0.0/8",
        "::1/128"
    ]
}
```

`GET /api/v1/servers/localhost/pool?name=pool-name`

New in version 1.6.1.

Get a quick overview of the pool named "pool-name".

Response JSON Object

- **`list`** – A list of metrics related to that pool
- **`servers`** (*`list`*) – A list of *`Server`* objects present in that pool

`GET /api/v1/servers/localhost/rings?maxQueries=NUM&maxResponses=NUM`

New in version 1.9.0.

Get the most recent queries and responses from the in-memory ring buffers. Returns up to `maxQueries` query entries if set, up to `maxResponses` responses if set, and the whole content of the ring buffers otherwise.

Response JSON Object

- **`queries`** (*`list`*) – The list of the most recent queries, as *`RingEntry`* objects
- **`responses`** (*`list`*) – The list of the most recent responses, as *`RingEntry`* objects

JSON Objects

`ConfigSetting`

An object representing a global configuration element. The following configuration are returned:

- `acl` The currently configured *ACLs*
- `control-socket` The currently configured *console address*
- `ecs-override`
- `ecs-source-prefix-v4` The currently configured *`setECSSourcePrefixV4()`*

- `ecs-source-prefix-v6` The currently configured *`setECSSourcePrefixV6()`*
- `fixup-case`
- `max-outstanding`
- `server-policy` The currently set *Loadbalancing and Server Policies*
- `stale-cache-entries-ttl`
- `tcp-recv-timeout`
- `tcp-send-timeout`
- `truncate-tc`
- `verbose`
- `verbose-health-checks` The currently configured *`setVerboseHealthChecks()`*

Object Properties

- **`name`** (*`string`*) – The name of the setting
- **`type`** (*`string`*) – "ConfigSetting"
- **`value`** (*`string`*) – The value for this setting

`DoHFrontend`

A description of a DoH bind dnsdist is listening on.

Object Properties

- **`bad-requests`** (*`integer`*) – Number of requests that could not be converted to a DNS query
- **`error-responses`** (*`integer`*) – Number of HTTP responses sent with a non-200 code
- **`get-queries`** (*`integer`*) – Number of DoH queries received via the GET HTTP method
- **`http-connects`** (*`integer`*) – Number of DoH TCP connections established to this frontend
- **`http1-queries`** (*`integer`*) – Number of DoH queries received over HTTP/1 (or connection attempts with a HTTP/1.1 ALPN when the nghttp2 provider is used)
- **`http1-x00-responses`** (*`integer`*) – Number of DoH responses sent, over HTTP/1, per response code (200, 400, 403, 500, 502)
- **`http1-other-responses`** (*`integer`*) – Number of DoH responses sent, over HTTP/1, with another response code
- **`http2-queries`** (*`integer`*) – Number of DoH queries received over HTTP/2
- **`http2-x00-responses`** (*`integer`*) – Number of DoH responses sent, over HTTP/2, per response code (200, 400, 403, 500, 502)
- **`http1-other-responses`** – Number of DoH responses sent, over HTTP/2, with another response code
- **`post-queries`** (*`integer`*) – Number of DoH queries received via the POST HTTP method
- **`redirect-responses`** (*`integer`*) – Number of HTTP redirect responses sent
- **`valid-responses`** (*`integer`*) – Number of valid DoH (2xx) responses sent

`Frontend`

A description of a bind dnsdist is listening on.

Object Properties

- **address** (*string*) – IP and port that is listened on
- **id** (*integer*) – Internal identifier
- **nonCompliantQueries** (*integer*) – Amount of non-compliant queries received by this frontend
- **queries** (*integer*) – The number of received queries on this bind
- **responses** (*integer*) – Amount of responses sent by this frontend
- **tcp** (*boolean*) – true if this is a TCP bind
- **tcpAvgConnectionDuration** (*integer*) – The average duration of a TCP connection (ms)
- **tcpAvgQueriesPerConnection** (*integer*) – The average number of queries per TCP connection
- **tcpClientTimeouts** (*integer*) – Amount of TCP connections terminated by a timeout while reading from the client
- **tcpCurrentConnections** (*integer*) – Amount of current incoming TCP connections from clients
- **tcpDiedReadingQuery** (*integer*) – Amount of TCP connections terminated while reading the query from the client
- **tcpDiedSendingResponse** (*integer*) – Amount of TCP connections terminated while sending a response to the client
- **tcpDownstreamTimeouts** (*integer*) – Amount of TCP connections terminated by a timeout while reading from the backend
- **tcpGaveUp** (*integer*) – Amount of TCP connections terminated after too many attempts to get a connection to the backend
- **tcpMaxConcurrentConnections** (*integer*) – Maximum number of concurrent incoming TCP connections from clients
- **tls10Queries** (*integer*) – Number of queries received by dnsdist over TLS 1.0
- **tls11Queries** (*integer*) – Number of queries received by dnsdist over TLS 1.1
- **tls12Queries** (*integer*) – Number of queries received by dnsdist over TLS 1.2
- **tls13Queries** (*integer*) – Number of queries received by dnsdist over TLS 1.3
- **tlsHandshakeFailuresDHKeyTooSmall** (*integer*) – Amount of TLS connections where the client has negotiated a not strong enough diffie-hellman key during the TLS handshake
- **tlsHandshakeFailuresInappropriateFallBack** (*integer*) – Amount of TLS connections where the client tried to negotiate an invalid, too old, TLS version
- **tlsHandshakeFailuresNoSharedCipher** (*integer*) – Amount of TLS connections were no cipher shared by both the client and the server could been found during the TLS handshake
- **tlsHandshakeFailuresUnknownCipher** (*integer*) – Amount of TLS connections where the client has tried to negotiate an unknown TLS cipher
- **tlsHandshakeFailuresUnknownKeyExchangeType** (*integer*) – Amount of TLS connections where the client has tried to negotiate an unknown TLS key-exchange mechanism
- **tlsHandshakeFailuresUnknownProtocol** (*integer*) – Amount of TLS connections where the client has tried to negotiate an unknown TLS version

- **tlsHandshakeFailuresUnsupportedEC** (*integer*) – Amount of TLS connections where the client has tried to negotiate an unsupported elliptic curve
- **tlsHandshakeFailuresUnsupportedProtocol** (*integer*) – Amount of TLS connections where the client has tried to negotiate a unsupported TLS version
- **tlsInactiveTicketKey** (*integer*) – Amount of TLS sessions resumed from an inactive key
- **tlsNewSessions** (*integer*) – Amount of new TLS sessions negotiated
- **tlsResumptions** (*integer*) – Amount of TLS sessions resumed
- **tlsUnknownQueries** (*integer*) – Number of queries received by dnsdist over an unknown TLS version
- **tlsUnknownTicketKey** (*integer*) – Amount of attempts to resume TLS session from an unknown key (possibly expired)
- **type** (*string*) – UDP, TCP, DoT or DoH
- **udp** (*boolean*) – true if this is a UDP bind

Pool

A description of a pool of backend servers.

Object Properties

- **id** (*integer*) – Internal identifier
- **cacheCleanupCount** (*integer*) – Number of times that cache was scanned for expired entries, or just to remove entries because it is full
- **cacheDeferredInserts** (*integer*) – The number of times an entry could not be inserted in the associated cache, if any, because of a lock
- **cacheDeferredLookups** (*integer*) – The number of times an entry could not be looked up from the associated cache, if any, because of a lock
- **cacheEntries** (*integer*) – The current number of entries in the associated cache, if any
- **cacheHits** (*integer*) – The number of cache hits for the associated cache, if any
- **cacheInsertCollisions** (*integer*) – The number of times an entry could not be inserted into the cache because a different entry with the same hash already existed
- **cacheLookupCollisions** (*integer*) – The number of times an entry retrieved from the cache based on the query hash did not match the actual query
- **cacheMisses** (*integer*) – The number of cache misses for the associated cache, if any
- **cacheSize** (*integer*) – The maximum number of entries in the associated cache, if any
- **cacheTTLTooShorts** (*integer*) – The number of times an entry could not be inserted into the cache because its TTL was set below the minimum threshold
- **name** (*string*) – Name of the pool
- **serversCount** (*integer*) – Number of backends in this pool

Rule

This represents a policy that is applied to queries

Object Properties

- **action** (*string*) – The action taken when the rule matches (e.g. "to pool abuse")
- **action-stats** (*dict*) – A list of statistics whose content varies depending on the kind of rule

- **`creationOrder`** (*`integer`*) – The order in which a rule has been created, mostly used for automated tools
- **`id`** (*`integer`*) – The position of this rule
- **`matches`** (*`integer`*) – How many times this rule was hit
- **`name`** (*`string`*) – The name assigned to this rule by the administrator, if any
- **`rule`** (*`string`*) – The matchers for the packet (e.g. "qname==bad-domain1.example., bad-domain2.example.")
- **`uuid`** (*`string`*) – The UUID of this rule

`ResponseRule`

This represents a policy that is applied to responses

Object Properties

- **`action`** (*`string`*) – The action taken when the rule matches (e.g. "drop")
- **`id`** (*`integer`*) – The identifier (or order) of this rule
- **`matches`** (*`integer`*) – How many times this rule was hit
- **`rule`** (*`string`*) – The matchers for the packet (e.g. "qname==bad-domain1.example., bad-domain2.example.")

`Server`

This object represents a backend server.

Object Properties

- **`address`** (*`string`*) – The remote IP and port
- **`id`** (*`integer`*) – Internal identifier
- **`latency`** (*`integer`*) – The current latency of this backend server for UDP queries, in milliseconds
- **`name`** (*`string`*) – The name of this server
- **`integer`** – nonCompliantResponses: Amount of non-compliant responses
- **`order`** (*`integer`*) – Order number
- **`outstanding`** (*`integer`*) – Number of currently outstanding queries
- **`pools`** (*`[string]`*) – The pools this server belongs to
- **`protocol`** (*`string`*) – The protocol used by this server (Do53, DoT, DoH)
- **`qps`** (*`integer`*) – The current number of queries per second to this server
- **`qpsLimit`** (*`integer`*) – The configured maximum number of queries per second
- **`queries`** (*`integer`*) – Total number of queries sent to this backend
- **`responses`** (*`integer`*) – Amount of responses received from this server
- **`reuseds`** (*`integer`*) – Number of queries for which a response was not received in time
- **`sendErrors`** (*`integer`*) – Number of network errors while sending a query to this server
- **`state`** (*`string`*) – The state of the server (e.g. "DOWN" or "up")
- **`tcpAvgConnectionDuration`** (*`integer`*) – The average duration of a TCP connection (ms)
- **`tcpAvgQueriesPerConnection`** (*`integer`*) – The average number of queries per TCP connection

- **tcpConnectTimeouts** (*integer*) – The number of TCP connect timeouts
- **tcpCurrentConnections** (*integer*) – The number of current TCP connections
- **tcpDiedReadingResponse** (*integer*) – The number of TCP I/O errors while reading the response
- **tcpDiedSendingQuery** (*integer*) – The number of TCP I/O errors while sending the query
- **tcpGaveUp** (*integer*) – The number of TCP connections failing after too many attempts
- **tcpLatency** (*integer*) – Server's latency when answering TCP questions in milliseconds
- **tcpMaxConcurrentConnections** (*integer*) – The maximum number of concurrent TCP connections
- **tcpNewConnections** (*integer*) – The number of established TCP connections in total
- **tcpReadTimeouts** (*integer*) – The number of TCP read timeouts
- **tcpReusedConnections** (*integer*) – The number of times a TCP connection has been reused
- **tcpTooManyConcurrentConnections** (*integer*) – Number of times we had to enforce the maximum number of concurrent TCP connections
- **tcpWriteTimeouts** (*integer*) – The number of TCP write timeouts
- **tlsResumptions** (*integer*) – The number of times a TLS session has been resumed
- **weight** (*integer*) – The weight assigned to this server
- **dropRate** (*float*) – The amount of packets dropped (timing out) per second by this server
- **healthCheckFailures** (*integer*) – Number of health check attempts that failed (total)
- **healthCheckFailureParsing** (*integer*) – Number of health check attempts that failed because the payload could not be parsed
- **healthCheckFailureTimeout** (*integer*) – Number of health check attempts that failed because the response was not received in time
- **healthCheckFailureNetwork** (*integer*) – Number of health check attempts that failed because of a network error
- **healthCheckFailureMismatch** (*integer*) – Number of health check attempts that failed because the ID, qname, qtype or qclass did not match
- **healthCheckFailureInvalid** (*integer*) – Number of health check attempts that failed because the DNS response was not valid

StatisticItem

This represents a statistics element.

Object Properties

- **name** (*string*) – The name of this statistic. See *Statistics*
- **type** (*string*) – "StatisticItem"
- **value** (*integer*) – The value for this item

RingEntry

This represents an entry in the in-memory ring buffers.

Object Properties

- **age** (*float*) – How long ago was the query or response received, in seconds
- **id** (*integer*) – The DNS ID
- **name** (*string*) – The requested domain name
- **requestor** (*string*) – The client IP and port
- **size** (*integer*) – The size of the query or response
- **qtype** (*integer*) – The requested DNS type
- **protocol** (*string*) – The DNS protocol the query or response was received over
- **rd** (*boolean*) – The RD flag
- **mac** (*string*) – The MAC address of the device sending the query
- **latency** (*float*) – The time it took for the response to be sent back to the client, in microseconds
- **rcode** (*int*) – The response code
- **tc** (*boolean*) – The TC flag
- **aa** (*boolean*) – The AA flag
- **answers** (*integer*) – The number of records in the answer section of the response
- **backend** (*string*) – The IP and port of the backend that returned the response, or “Cache” if it was a cache-hit

17.2 Server pools

dnsdist has the concept to “server pools”, any number of servers can belong to a group. A default pool, identified by the empty string `''` is always present, and *newServer* without a pool argument will assign the new server to that pool.

Let’s say we know we’re getting a whole bunch of traffic for a domain used in DoS attacks, for example ‘example.com’. We can do two things with this kind of traffic. Either we block it outright, like this:

```
addAction("bad-domain.example.", DropAction())
```

Or we configure a server pool dedicated to receiving the nasty stuff:

```
newServer({address="192.0.2.3", pool="abuse"})         -- Add a backend server␣
↪with address 192.0.2.3 and assign it to the "abuse" pool
addAction({'bad-domain1.example', 'bad-domain2.example.'}, PoolAction("abuse")) --␣
↪Send all queries for "bad-domain1.example." and "bad-domain2.example" to the
↪"abuse" pool
```

The wonderful thing about this last solution is that it can also be used for things where a domain might possibly be legit, but it is still causing load on the system and slowing down the internet for everyone. With such an abuse server, ‘bad traffic’ still gets a chance of an answer, but without impacting the rest of the world (too much).

We can similarly add clients to the abuse server:

```
addAction({"192.168.12.0/24", "192.168.13.14"}, PoolAction("abuse"))
```

To define a pool that should receive only a *QPS*-limited amount of traffic, do:

```
addAction("com.", QPSPoolAction(10000, "gtld-cluster"))
```

Traffic exceeding the *QPS* limit will not match that rule, and subsequent rules will apply normally.

`Servers` can be added to or removed from pools with the `Server:addPool()` and `Server:rmPool()` functions respectively:

```
getServer(4):addPool("abuse")
getServer(4):rmPool("abuse")
```

17.3 Loadbalancing and Server Policies

`dnsdist` selects the server (if there are multiple eligible) to send queries to based on the configured policy. Only servers that are marked as 'up', either forced so by the administrator or as the result of the last health check, might be selected.

17.3.1 Built-in Policies

`leastOutstanding`

The default load balancing policy is called `leastOutstanding`, which means the server with the least queries 'in the air' is picked. The exact selection algorithm is:

- pick the server with the least queries 'in the air' ;
- in case of a tie, pick the one with the lowest configured 'order' ;
- in case of a tie, pick the one with the lowest measured latency (over an average on the last 128 queries answered by that server).

`firstAvailable`

The `firstAvailable` policy, picks the first available server that has not exceeded its QPS limit, ordered by increasing 'order'. If all servers are above their QPS limit, a server is selected based on the `leastOutstanding` policy. For now this is the only policy using the QPS limit.

`wrandom`

A further policy, `wrandom` assigns queries randomly, but based on the weight parameter passed to `newServer()`.

For example, if two servers are available, the first one with a weight of 2 and the second one with a weight of 1 (the default), the first one should get two-thirds of the incoming queries and the second one the remaining third.

Since 1.5.0, a bounded-load version is also supported, trying to prevent one server from receiving much more queries than intended, even if the distribution of queries is not perfect. This "weighted random with bounded loads" algorithm is enabled by setting `setWeightedBalancingFactor()` to a value other than 0, which is the default. This value is the maximum number of outstanding queries that a given server can have at a given time, as a ratio of the total number of outstanding queries for all the active servers in the pool, pondered by the weight of the server.

The algorithm will try to select a server randomly, as is done when no bounded-load is set, but will disqualify all servers that have more outstanding queries than intended times the factor, until a suitable server is found. The higher the factor, the more imbalance between the servers is allowed.

For example, if we have two servers, with respective weights of 1 and 4, we expect the first server to get a fifth of the queries, and the second one 4/5. As the random distribution is not perfect, some server might get more queries than expected. Setting `setWeightedBalancingFactor()` to 1.1 limits the imbalance between the ratio of outstanding queries actually handled by a server and the expected number, so in this example the first server would not be allowed to handle more than 1.1/5 of all the outstanding queries at a given time.

whashed

whashed is a similar weighted policy, but assigns questions with identical hash to identical servers, allowing for better cache concentration ('sticky queries'). The current hash algorithm is based on the qname of the query.

setWHashedPertubation(*value*)

Set the hash perturbation value to be used in the whashed policy instead of a random one, allowing to have consistent whashed results on different instances.

Since 1.5.0, a bounded-load version is also supported, trying to prevent one server from receiving much more queries than intended, even if the distribution of queries is not perfect. This "weighted hashing with bounded loads" algorithm is enabled by setting *setWeightedBalancingFactor()* to a value other than 0, which is the default. This value is the maximum number of outstanding queries that a given server can have at a given time, as a ratio of the total number of outstanding queries for all the active servers in the pool, pondered by the weight of the server.

The algorithm will try to select a server based on the hash of the qname, as is done when no bounded-load is set, but will disqualify all servers that have more outstanding queries than intended times the factor, until a suitable server is found. The higher the factor, the more imbalance between the servers is allowed.

For example, if we have two servers, with respective weights of 1 and 4, we expect the first server to get a fifth of the queries, and the second one 4/5. If the qname of the queries are not perfectly distributed, some server might get more queries than expected. Setting *setWeightedBalancingFactor()* to 1.1 limits the imbalance between the ratio of outstanding queries actually handled by a server and the expected number, so in this example the first server would not be allowed to handle more than 1.1/5 of all the outstanding queries at a given time.

chashed

chashed is a consistent hashing distribution policy. Identical questions with identical hashes will be distributed to the same servers. But unlike the whashed policy, this distribution will keep consistent over time. Adding or removing servers will only remap a small part of the queries.

Increasing the weight of servers to a value larger than the default is required to get a good distribution of queries. Small values like 100 or 1000 should be enough to get a correct distribution. This is a side-effect of the internal implementation of the consistent hashing algorithm, which assigns as many points on a circle to a server than its weight, and distributes a query to the server who has the closest point on the circle from the hash of the query's qname. Therefore having very few points, as is the case with the default weight of 1, leads to a poor distribution of queries.

You can also set the hash perturbation value, see *setWHashedPertubation()*. To achieve consistent distribution over **dnsdist** restarts, you will also need to explicitly set the backend's UUIDs with the id option of *newServer()*. You can get the current UUIDs of your backends by calling *showServers()* with the showUUIDs=true option.

Since 1.5.0, a bounded-load version is also supported, preventing one server from receiving much more queries than intended, even if the distribution of queries is not perfect. This "consistent hashing with bounded loads" algorithm is enabled by setting *setConsistentHashingBalancingFactor()* to a value other than 0, which is the default. This value is the maximum number of outstanding queries that a given server can have at a given time, as a ratio of the total number of outstanding queries for all the active servers in the pool, pondered by the weight of the server.

The algorithm will try to select a server based on the hash of the qname, as is done when no bounded-load is set, but will disqualify all servers that have more outstanding queries than intended times the factor, until a suitable server is found. The higher the factor, the more imbalance between the servers is allowed.

For example, if we have two servers, with respective weights of 1 and 4, we expect the first server to get a fifth of the queries, and the second one 4/5. If the qname of the queries are not perfectly distributed, some server might get more queries than expected. Setting *setConsistentHashingBalancingFactor()* to 1.1 limits the imbalance between the ratio of outstanding queries actually handled by a server and the expected number, so in this example the first server would not be allowed to handle more than 1.1/5 of all the outstanding queries at a given time.

roundrobin

The last available policy is `roundrobin`, which indiscriminately sends each query to the next server that is up. If all servers are down, the policy will still select one server by default. Setting *setRoundRobinFailOnNoServer()* to `true` will change this behavior.

17.3.2 Lua server policies

If you don't like the default policies you can create your own, like this for example:

```
counter=0
function luaroundrobin(servers, dq)
     counter=counter+1
     return servers[1+(counter % #servers)]
end

setServerPolicyLua("luaroundrobin", luaroundrobin)
```

Incidentally, this is similar to setting: `setServerPolicy(roundrobin)` which uses the C++ based roundrobin policy.

Or:

```
newServer("192.168.1.2")
newServer({address="8.8.4.4", pool="numbered"})

function splitSetup(servers, dq)
  if(string.match(dq.qname:toString(), "%d"))
  then
    print("numbered pool")
    return leastOutstanding.policy(getPoolServers("numbered"), dq)
  else
    print("standard pool")
    return leastOutstanding.policy(servers, dq)
  end
end

setServerPolicyLua("splitsetup", splitSetup)
```

A faster, FFI version is also available since 1.5.0:

```
local ffi = require("ffi")
local C = ffi.C

local counter = 0
function luaffiroundrobin(servers_list, dq)
  counter = counter + 1
  return (counter % tonumber(C.dnsdist_ffi_servers_list_get_count(servers_list)))
end
setServerPolicyLuaFFI("luaffiroundrobin", luaffiroundrobin)
```

Note that this version returns the index (starting at 0) of the server to select, instead of returning the server itself. It was initially not possible to indicate that all servers were unavailable from these policies, but since 1.9.2 returning a value equal or greater than the number of servers will be interpreted as such.

For performance reasons, 1.6.0 introduced per-thread Lua FFI policies that are run in a lock-free per-thread Lua context instead of the global one. This reduces contention between threads at the cost of preventing sharing data between threads for these policies. Since the policy needs to be recompiled in the context of each thread instead of the global one, Lua code that returns a function should be passed to the function as a string instead of directly passing the name of a function:

```
setServerPolicyLuaFFIPerThread("luaffiroundrobin", [[
  local ffi = require("ffi")
  local C = ffi.C

  local counter = 0
  return function(servers_list, dq)
    counter = counter + 1
    return (counter % tonumber(C.dnsdist_ffi_servers_list_get_count(servers_list)))
  end
]])
```

Note that this version, like the one above, returns the index (starting at 0) of the server to select. It was initially not possible to indicate that all servers were unavailable from these policies, but since 1.9.2 returning a value equal or greater than the number of servers will be interpreted as such.

17.3.3 ServerPolicy Objects

class ServerPolicy

This represents a server policy. The built-in policies are of this type

`ServerPolicy.`**`policy`**(*servers*, *dq*) → Server

Run the policy to receive the server it has selected.

Parameters

- **servers** – A list of *Server* objects
- **dq** (DNSQuestion) – The incoming query

`ServerPolicy.`**`ffipolicy`**

For policies implemented using the Lua FFI interface, the policy function itself.

`ServerPolicy.`**`isFFI`**

Whether a Lua-based policy is implemented using the FFI interface.

`ServerPolicy.`**`isLua`**

Whether this policy is a native (C++) policy or a Lua-based one.

`ServerPolicy.`**`isPerThread`**

Whether a FFI Lua-based policy is executed in a lock-free per-thread context instead of running in the global Lua context.

`ServerPolicy.`**`name`**

The name of the policy.

`ServerPolicy.`**`policy`**

The policy function itself, except for FFI policies.

`Server:`**`toString`**()

Return a textual representation of the policy.

17.3.4 Functions

`newServerPolicy`(*name*, *function*) → ServerPolicy

Create a policy object from a Lua function. `function` must match the prototype for *ServerPolicy.policy()*.

Parameters

- **name** (*string*) – Name of the policy
- **function** (*string*) – The function to call for this policy

setConsistentHashingBalancingFactor (*factor*)

Set the maximum imbalance between the number of outstanding queries intended for a given server, based on its weight, and the actual number, when using the `chashed` consistent hashing load-balancing policy. Default is 0, which disables the bounded-load algorithm.

setServerPolicy (*policy*)

Set server selection policy to `policy`.

Parameters **policy** (`ServerPolicy`) – The policy to use

setServerPolicyLua (*name*, *function*)

Set server selection policy to one named `name` and provided by `function`.

Parameters

- **name** (*string*) – name for this policy
- **function** (*string*) – name of the function

setServerPolicyLuaFFI (*name*, *function*)

New in version 1.5.0.

Changed in version 1.9.2: Returning a value equal or greater than the number of servers will be interpreted as all servers being unavailable.

Set server selection policy to one named `name` and provided by the FFI function `function`.

Parameters

- **name** (*string*) – name for this policy
- **function** (*string*) – name of the FFI function

setServerPolicyLuaFFIPerThread (*name*, *code*)

New in version 1.6.0.

Changed in version 1.9.2: Returning a value equal or greater than the number of servers will be interpreted as all servers being unavailable.

Set server selection policy to one named `name` and the Lua FFI function returned by the Lua code passed in `code`. The resulting policy will be executed in a lock-free per-thread context, instead of running in the global Lua context.

Parameters

- **name** (*string*) – name for this policy
- **code** (*string*) – Lua FFI code returning the function to execute as a server selection policy

setServFailWhenNoServer (*value*)

If set, return a ServFail when no servers are available, instead of the default behaviour of dropping the query.

Parameters **value** (*bool*) – whether to return a servfail instead of dropping the query

setPoolServerPolicy (*policy*, *pool*)

Set the server selection policy for `pool` to `policy`.

Parameters

- **policy** (`ServerPolicy`) – The policy to apply
- **pool** (*string*) – Name of the pool

setPoolServerPolicyLua (*name*, *function*, *pool*)

Set the server selection policy for `pool` to one named `name` and provided by `function`.

Parameters

- **name** (*string*) – name for this policy
- **function** (*string*) – name of the function

- **pool** (*string*) – Name of the pool

setRoundRobinFailOnNoServer (*value*)

New in version 1.4.0.

By default the roundrobin load-balancing policy will still try to select a backend even if all backends are currently down. Setting this to true will make the policy fail and return that no server is available instead.

Parameters **value** (*bool*) – whether to fail when all servers are down

setWeightedBalancingFactor (*factor*)

Set the maximum imbalance between the number of outstanding queries intended for a given server, based on its weight, and the actual number, when using the `whashed` or `wrandom` load-balancing policy. Default is 0, which disables the bounded-load algorithm.

showPoolServerPolicy (*pool*)

Print server selection policy for `pool`.

Parameters **pool** (*string*) – The pool to print the policy for

CHAPTER

EIGHTEEN

ADVANCED TOPICS

These chapters contain information on the advanced features of dnsdist

18.1 Access Control

dnsdist can be used to front traditional recursive nameservers, these usually come with a way to limit the network ranges that may query it to prevent becoming an *open resolver*. To be a good internet citizen, dnsdist by default listens on the loopback address (*127.0.0.1:53*) and limits queries to these loopback, **RFC 1918** and other local addresses:

- `127.0.0.0/8`
- `10.0.0.0/8`
- `100.64.0.0/10`
- `169.254.0.0/16`
- `192.168.0.0/16`
- `172.16.0.0/12`
- `::1/128`
- `fc00::/7`
- `fe80::/10`

The ACL applies to queries received over UDP, TCP, DNS over TLS and DNS over HTTPS.

Further more, dnsdist only listens for queries on the local-loopback interface by default.

18.1.1 Listening on different addresses

To listen on other addresses than just the local addresses, use `setLocal()` and `addLocal()`.

`setLocal()` **resets** the list of current listen addresses to the specified address and `addLocal()` adds an additional listen address. To listen on `127.0.0.1:5300`, `192.0.2.1:53` and UDP-only on `[2001:db8::15::47]:53`, configure the following:

```
setLocal('127.0.0.1:5300')
addLocal('192.0.2.1') -- Port 53 is default is none is specified
addLocal('2001:db8::15::47', false)
```

Listen addresses cannot be modified at runtime and must be specified in the configuration file.

As dnsdist is IPv4 and IPv6 agnostic, this means that dnsdist internally does not know the difference. So feel free to listen on the magic `0.0.0.0` or `::` addresses, dnsdist does the right thing to set the return address of queries, but set your *ACL* properly.

18.1.2 Modifying the ACL

ACLs can be modified at runtime from the *Working with the dnsdist Console*. To inspect the currently active *ACL*, run `showACL()`.

To add a new network range to the existing ACL, use `addACL()`:

```
addACL('192.0.2.0/25')
addACL('2001:db8::1') -- No netmask specified, only allow this address
```

To remove a previously added network range from the existing ACL, use `rmACL()`:

```
rmACL('192.0.2.0/25')
rmACL('2001:db8::1') -- No netmask specified, only remove this address
```

dnsdist also has the `setACL()` function that accepts a list of netmasks and resets the ACL to that list:

```
setACL({'192.0.2.0/25', '2001:db8:15::bea/64'})
```

To set the ACL from a file containing a list of netmasks, use `setACLFromFile()`:

```
setACLFromFile('/etc/dnsdist/query.acl')
```

18.2 Passing the source address to the backend

dnsdist, as a load-balancer, receives the UDP datagrams and terminates the TCP connections with the client. It therefore knows the source IP address and port of that client, as well as the original destination address, port, and protocol. Very often the backend needs to know that information as well, to pass EDNS Client Subnet to an authoritative server, to do GeoIP-based processing or even custom filtering.

There are several ways to pass that information using dnsdist: EDNS Client Subnet, X-Proxied-For and the Proxy Protocol.

18.2.1 Using EDNS Client Subnet

EDNS Client Subnet (ECS) is a standardized EDNS option designed to pass a bit of information about the client from a resolver to authoritative servers. While it was not designed with our use-case in mind, it can be used by dnsdist to send the source IP, but only the source IP, to its backend.

In order to provide the downstream server with the address of the real client, or at least the one talking to dnsdist, the `useClientSubnet` parameter can be used when creating a `new server`. This parameter indicates whether an EDNS Client Subnet option should be added to the request.

The default source prefix-length is 24 for IPv4 and 56 for IPv6, meaning that for a query received from 192.0.2.42, the EDNS Client Subnet value sent to the backend will be 192.0.2.0. This can be changed with `setECSSourcePrefixV4()` and `setECSSourcePrefixV6()`.

If the incoming request already contains an EDNS Client Subnet value, it will not be overridden unless `setECSOverride()` is set to `true`.

In addition to the global settings, rules and Lua bindings can alter this behavior per query:

- calling `SetDisableECSAction()` or setting `dq.useECS` to `false` prevents the sending of the ECS option.
- calling `SetECSOverrideAction()` or setting `dq.ecsOverride` will override the global `setECSOverride()` value.
- calling `SetECSPrefixLengthAction(v4, v6)()` or setting `dq.ecsPrefixLength` will override the global `setECSSourcePrefixV4()` and `setECSSourcePrefixV6()` values.

In effect this means that for the EDNS Client Subnet option to be added to the request, `useClientSubnet` should be set to `true` for the backend used (default to `false`) and ECS should not have been disabled by calling *`SetDisableECSAction()`* or setting `dq.useECS` to `false` (default to true).

Note that any trailing data present in the incoming query is removed when an OPT (or XPF) record has to be inserted.

In addition to the drawback that it can only pass the source IP address, and the fact that it needs to override any existing ECS option, adding that option requires parsing and editing the query, as well as parsing and editing the response in most cases.

Payload	Required processing
Query, no EDNS	add an OPT record
Query, EDNS without ECS	edit the OPT record to add an ECS option
Query, ECS	edit the OPT record to overwrite the ECS option
Response, no EDNS	none
Response, EDNS without ECS	remove the OPT record if needed
Response, EDNS with ECS	remove or edit the ECS option if needed

18.2.2 X-Proxied-For

Note: This is a deprecated feature that will be removed in the near future.

The experimental XPF record (from draft-bellis-dnsop-xpf) is an alternative to the use of EDNS Client Subnet which has the advantages of preserving any existing EDNS Client Subnet value sent by the client, and of passing along the original destination address, as well as the initial source and destination ports.

In order to provide the downstream server with the address of the real client, or at least the one talking to dnsdist, the `addXPF` parameter can be used when creating a *`new server`*. This parameter indicates whether an XPF record shall be added to the query. Since that record is experimental, there is currently no option code assigned to it, and therefore one needs to be specified as an argument to the `addXPF` parameter.

If the incoming request already contains a XPF record, it will not be overwritten. Instead a new one will be added to the query and the existing one will be preserved. That might be an issue by allowing clients to spoof their source address by adding a forged XPF record to their query. That can be prevented by using a rule to drop incoming queries containing a XPF record (in that example the 65280 option code has been assigned to XPF):

```
addAction(RecordsTypeCountRule(DNSSection.Additional, 65280, 1, 65535),␣
↪DropAction())
```

18.2.3 Proxy Protocol

The Proxy Protocol has been designed by the HAProxy folks for HTTP over TCP, but is generic enough to be used in other places, and is a de-facto standard with implementations in nginx and postfix, for example. It works by pre-pending a small header at the very beginning of a UDP datagram or TCP connection, which holds the initial source and destination addresses and ports, and can also contain several custom values in a Type-Length-Value format. More information about the Proxy Protocol can be found at https://www.haproxy.org/download/2.2/doc/proxy-protocol.txt

In order to use it in dnsdist, the `useProxyProtocol` parameter can be used when creating a *`new server`*. This parameter indicates whether a Proxy Protocol version 2 (binary) header should be prepended to the query before forwarding it to the backend, over UDP or TCP. Such a Proxy Protocol header can also be passed from the client to dnsdist, using *`setProxyProtocolACL()`* to specify which clients to accept it from. Note that a proxy protocol payload will be required from these clients, regular DNS queries will no longer be accepted if they are not preceded by a proxy protocol payload.

If `setProxyProtocolApplyACLToProxiedClients()` is set (default is false), the general ACL will be applied to the source IP address as seen by dnsdist first, but also to the source IP address provided in the Proxy Protocol header.

Custom values can be added to the header via `DNSQuestion:addProxyProtocolValue()`, `DNSQuestion:setProxyProtocolValues()`, `SetAdditionalProxyProtocolValueAction()` and `SetProxyProtocolValuesAction()`. Be careful that Proxy Protocol values are sent once at the beginning of the TCP connection for TCP and DoT queries. That means that values received on an incoming TCP connection will be inherited by subsequent queries received over the same incoming TCP connection, if any, but values set to a query will not be inherited by subsequent queries. Please also note that the maximum size of a Proxy Protocol header dnsdist is willing to accept is 512 bytes by default, although it can be set via `setProxyProtocolMaximumPayloadSize()`.

dnsdist 1.5.0 only supports outgoing Proxy Protocol. Support for parsing incoming Proxy Protocol headers has been implemented in 1.6.0, except for DoH where it does not make sense anyway, since HTTP headers already provide a mechanism for that.

Both the PowerDNS Authoritative Server and the Recursor can parse PROXYv2 headers, if configured to do so with their *proxy-protocol-from* setting.

18.2.4 Influence on caching

When dnsdist's packet cache is in use, it is important to note that the cache lookup is done **after** adding ECS, because it prevents serving the same response to clients from different subnets when ECS is passed to an authoritative server doing GeoIP, or to a backend doing custom filtering. However that means that passing a narrow ECS source will effectively kill dnsdist's cache ratio, since a given answer will only be a cache hit for clients in the same ECS subnet. Therefore, unless a broad ECS source (greater than 24, for example) is used, it's better to disable caching.

One exception to that rule is the zero-scope feature, which allows dnsdist to detect that a response sent by the backend has a 0-scope ECS value, indicating that the answer is not ECS-specific and can be used for all clients. dnsdist will then store the answer in its packet cache using the initial query, before ECS has been added. For that feature to work, dnsdist will look up twice into the packet cache when a query arrives, first without and then with ECS. That way, when most of the responses sent by a backend are not ECS-specific and can be served to all clients, dnsdist will still be able to have a great cache-hit ratio for non ECS-specific entries.

That feature is enabled by setting `disableZeroScope=false` on `newServer()` (default) and `parseECS=true` on `newPacketCache()` (not the default).

Things are different for XPF and the proxy protocol, because dnsdist then does the cache lookup **before** adding the payload. It means that caching can still be enabled as long as the response is not source-dependent, but should be disabled otherwise.

Protocol	Standard	Require DNS parsing	Contains ports	Caching
ECS	Yes	Query and response	No	Only with broad source
ECS (zero-scope)	Yes	Query and response	No	Yes
XPF	No	Query	Yes	Depends on the backend
Proxy Protocol	No	No	Yes	Depends on the backend

18.3 TeeAction: copy the DNS traffic stream

This action sends off a copy of a UDP query to another server, and keeps statistics on the responses received. Sample use:

```
> addAction(AllRule(), TeeAction("192.0.2.54"))
> getAction(0):printStats()
refuseds    0
nxdomains   0
```

(continues on next page)

(continued from previous page)

```
noerrors    0
servfails   0
recv-errors 0
tcp-drops   0
responses   0
other-rcode 0
send-errors 0
queries 0
```

It is also possible to share a `TeeAction()` between several rules. Statistics will be combined in that case.

18.4 Lua actions in rules

While we can pass every packet through the `blockFilter()` functions, it is also possible to configure **dnsdist** to only hand off some packets for Lua inspection. If you think Lua is too slow for your query load, or if you are doing heavy processing in Lua, this may make sense.

To select specific packets for Lua attention, use `addAction()` with `LuaAction()`, or `addResponseAction()` with `LuaResponseAction()`.

A sample configuration could look like this:

```
function luarule(dq)
  if(dq.qtype==35) -- NAPTR
  then
    return DNSAction.Pool, "abuse" -- send to abuse pool
  else
    return DNSAction.None, ""      -- no action
  end
end

addAction(AllRule(), LuaAction(luarule))
```

18.5 Runtime-modifiable IP address sets

From within `maintenance()` or other places, we may find that certain IP addresses must be treated differently for a certain time.

This may be used to temporarily shunt traffic to another pool for example.

`TimedIPSetRule()` creates an object to which native IP addresses can be added in `ComboAddress` form.

TimedIPSetRule() → TimedIPSetRule

Returns a `TimedIPSetRule`.

class TimedIPSetRule

Can be used to handle IP addresses differently for a certain time.

:add(*address*, *seconds*)

Add an IP address to the set for the next `second` seconds.

Parameters

- **address** (`ComboAddress`) – The address to add
- **seconds** (*int*) – Time to keep the address in the Rule

:cleanup()

Purge the set from expired IP addresses

:clear()
: Clear the entire set

:slice()
: Convert the TimedIPSetRule into a DNSRule that can be passed to *addAction()*

A working example:

```
tisrElGoog=TimedIPSetRule()
tisrRest=TimedIPSetRule()
addAction(tisrElGoog:slice(), PoolAction("elgoog"))
addAction(tisrRest:slice(), PoolAction(""))

elgoogPeople=newNMG()
elgoogPeople:addMask("192.168.5.0/28")

function pickPool(dq)
        if(elgoogPeople:match(dq.remoteaddr)) -- in real life, this would be external
        then
                print("Lua caught query for a googlePerson")
                tisrElGoog:add(dq.remoteaddr, 10)
                return DNSAction.Pool, "elgoog"
        else
                print("Lua caught query for restPerson")
                tisrRest:add(dq.remoteaddr, 60)
                return DNSAction.None, ""
        end
end

addAction(AllRule(), LuaAction(pickPool))
```

18.6 Rules for traffic exceeding QPS limits

Traffic that exceeds a QPS limit, in total or per IP (subnet) can be matched by the *MaxQPSIPRule()*-rule. For example:

```
addAction(MaxQPSIPRule(5, 32, 48), DelayAction(100))
```

This measures traffic per IPv4 address and per /48 of IPv6, and if UDP traffic for such an address (range) exceeds 5 *qps*, it gets delayed by 100ms.

As another example:

```
addAction(MaxQPSIPRule(5), SetNoRecurseAction())
```

This strips the Recursion Desired (RD) bit from any traffic per IPv4 or IPv6 /64 that exceeds 5 qps. This means any those traffic bins is allowed to make a recursor do ‘work’ for only 5 qps.

If this is not enough, try:

```
addAction(MaxQPSIPRule(5), DropAction())
-- or
addAction(MaxQPSIPRule(5), TCAction())
```

This will respectively drop traffic exceeding that 5 QPS limit per IP or range, or return it with TC=1, forcing clients to fall back to TCP.

To turn this per IP or range limit into a global limit, use `NotRule(MaxQPSRule(5000))` instead of *MaxQPSIPRule()*.

18.7 eBPF Socket Filtering

dnsdist can use eBPF socket filtering on recent Linux kernels (4.1+) built with eBPF support (CONFIG_BPF, CONFIG_BPF_SYSCALL, ideally CONFIG_BPF_JIT). It requires dnsdist to have the CAP_SYS_ADMIN capabilities at startup, or the more restrictive CAP_BPF one since Linux 5.8.

Note: To retain the required capability, CAP_SYS_ADMIN or CAP_BPF depending on the Linux kernel version, it is necessary to call *addCapabilitiesToRetain()* during startup, as **dnsdist** drops capabilities after startup.

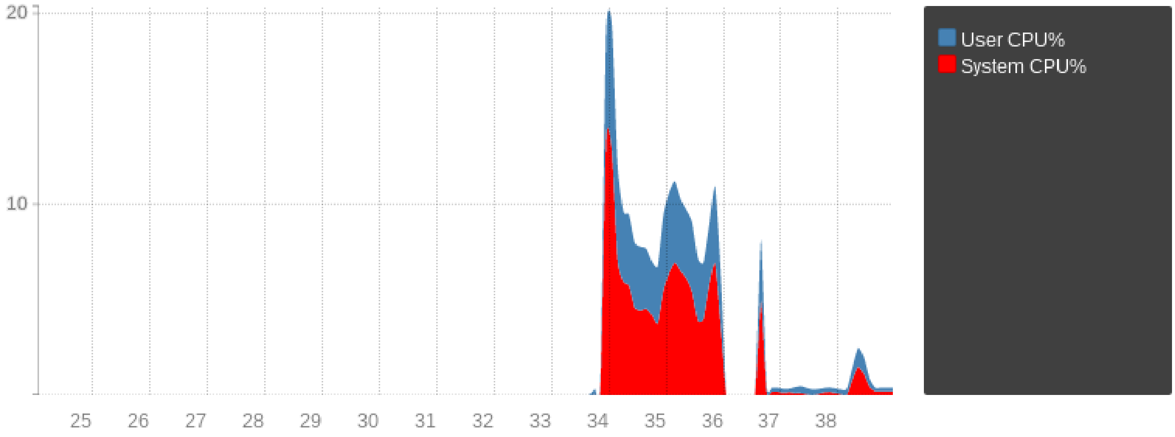
Note: eBPF can be used by unprivileged users lacking the CAP_SYS_ADMIN (or CAP_BPF) capability on some kernels, depending on the value of the kernel.unprivileged_bpf_disabled sysctl. Since 5.15 that kernel build setting BPF_UNPRIV_DEFAULT_OFF is enabled by default, which prevents unprivileged users from using eBPF.

Note: AppArmor users might need to update their policy to allow dnsdist to keep the CAP_SYS_ADMIN (or CAP_BPF) capability. Adding a capability bpf, (for CAP_BPF) line to the policy file is usually enough.

Note: In addition to keeping the correct capability, large maps might require an increase of RLIMIT_MEMLOCK, as mentioned below.

This feature allows dnsdist to ask the kernel to discard incoming packets in kernel-space instead of them being copied to userspace just to be dropped, thus being a lot of faster. The current implementation supports dropping UDP and TCP queries based on the source IP and UDP datagrams on exact DNS names. We have not been able to implement suffix matching yet, due to a limit on the maximum number of EBPF instructions.

The following figure show the CPU usage of dropping around 20k qps of traffic, first in userspace (34 to 36) then in kernel space with eBPF (37 to 39). The spikes are caused because the drops are triggered by dynamic rules, so the first spike is the abuse traffic before a rule is automatically inserted, and the second spike is because the rule expires automatically after 60s before being inserted again.



The BPF filter can be used to block incoming queries manually:

```
> bpf = newBPFFilter({ipv4MaxItems=1024, ipv6MaxItems=1024, qnamesMaxItems=1024})
> bpf:attachToAllBinds()
> bpf:block(newCA("2001:DB8::42"))
> bpf:blockQName(newDNSName("evildomain.com"), 255)
```

(continues on next page)

(continued from previous page)

```
> bpf:getStats()
[2001:DB8::42]: 0
evildomain.com. 255: 0
> bpf:unblock(newCA("2001:DB8::42"))
> bpf:unblockQName(newDNSName("evildomain.com"), 255)
> bpf:getStats()
```

The `BPFFilter:blockQName()` method can be used to block queries based on the exact qname supplied, in a case-insensitive way, and an optional qtype. Using the 255 (ANY) qtype will block all queries for the qname, regardless of the qtype. Contrary to source address filtering, qname filtering only works over UDP. TCP qname filtering can be done the usual way:

```
addAction(AndRule({TCPRule(true), makeRule("evildomain.com")}), DropAction())
```

The `BPFFilter:attachToAllBinds()` method attaches the filter to every existing bind at runtime. It cannot use at configuration time. The `setDefaultBPFFilter()` should be used at configuration time.

The `BPFFilter:attachToAllBinds()` automatically attached to every bind:

```
bpf = newBPFFilter({ipv4MaxItems=1024, ipv6MaxItems=1024, qnamesMaxItems=1024})
setDefaultBPFFilter(bpf)
```

Finally, it's also possible to attach it to specific binds at runtime:

```
> bpf = newBPFFilter({ipv4MaxItems=1024, ipv6MaxItems=1024, qnamesMaxItems=1024})
> showBinds()
#   Address              Protocol  Queries
0   [::]:53              UDP       0
1   [::]:53              TCP       0
> bd = getBind(0)
> bd:attachFilter(bpf)
```

dnsdist also supports adding dynamic, expiring blocks to a BPF filter:

```
bpf = newBPFFilter({ipv4MaxItems=1024, ipv6MaxItems=1024, qnamesMaxItems=1024})
setDefaultBPFFilter(bpf)
local dbr = dynBlockRulesGroup()
dbr:setQueryRate(20, 10, "Exceeded query rate", 60)

function maintenance()
  dbr:apply()
end
```

This will dynamically block all hosts that exceeded 20 queries/s as measured over the past 10 seconds, and the dynamic block will last for 60 seconds.

Since 1.6.0, the default BPF filter set via `setDefaultBPFFilter()` will automatically get used when a "drop" dynamic block is inserted via a *DynBlockRulesGroup*, which provides a better way to combine dynamic blocks with eBPF filtering. Before that, it was possible to use the `addBPFFilterDynBlocks()` method instead:

```
-- this is a legacy method, please see above for DNSdist >= 1.6.0
bpf = newBPFFilter({ipv4MaxItems=1024, ipv6MaxItems=1024, qnamesMaxItems=1024})
setDefaultBPFFilter(bpf)
dbpf = newDynBPFFilter(bpf)
function maintenance()
        addBPFFilterDynBlocks(exceedQRate(20, 10), dbpf, 60)
        dbpf:purgeExpired()
end
```

The dynamic eBPF blocks and the number of queries they blocked can be seen in the web interface and retrieved from the API. Note however that eBPF dynamic objects need to be registered before they appear in the web interface or the API, using the *registerDynBPFFilter()* function:

```
registerDynBPFFilter(dbpf)
```

They can be unregistered at a later point using the *unregisterDynBPFFilter()* function. Since 1.8.2, the metrics for the BPF filter registered via *setDefaultBPFFilter()* are exported as well.

18.7.1 Requirements

In addition to the capabilities explained above, that feature might require an increase of the memory limit associated to a socket, via the sysctl setting `net.core.optmem_max`. When attaching an eBPF program to a socket, the size of the program is checked against this limit, and the default value might not be enough.

Large map sizes might also require an increase of `RLIMIT_MEMLOCK`, which can be done by adding `LimitMEMLOCK=limit` in the systemd unit file, where limit is specified using byte as unit. It can also be done manually for testing purposes, in a non-permanent way, by using `ulimit -l`.

To change the default hard limit on `RLIMIT_MEMLOCK` add the following line to `/etc/security/limits.conf` for the

> $USER hard memlock 1024

18.7.2 External program, maps and XDP filtering

Since 1.7.0 dnsdist has the ability to expose its eBPF map to external programs. That feature makes it possible to populate the client IP addresses and qnames maps from dnsdist, usually using the dynamic block mechanism, and to act on the content of these maps from an external program, including a XDP one. For example, to instruct dnsdist to create under the `/sys/fs/bpf` mount point of type `bpf` three maps of maximum 1024 entries each, respectively pinned to `/sys/fs/bpf/dnsdist/addr-v4`, `/sys/fs/bpf/dnsdist/addr-v6`, `/sys/fs/bpf/dnsdist/qnames` for IPv4 addresses, IPv6 ones, and qnames:

```
bpf = newBPFFilter({maxItems=1024, pinnedPath='/sys/fs/bpf/dnsdist/addr-v4'},
↪{maxItems=1024, pinnedPath='/sys/fs/bpf/dnsdist/addr-v6'}, {maxItems=1024,␣
↪pinnedPath='/sys/fs/bpf/dnsdist/qnames'}, true)
```

Note: By default only root can write into a bpf mount point, but it is possible to create a `dnsdist/` sub-directory with `mkdir` and to make it owned by the `dnsdist` user with `chown`.

The last parameter to *newBPFFilter()* is set to `true` to indicate to dnsdist not to load its internal eBPF socket filter program, which is not needed since packets will be intercepted by an external program and would at best duplicate the work done by the other program. It also tell dnsdist to use a slightly different format for the eBPF maps:

- IPv4 and IPv6 maps still use the address as key, but the value contains an action field in addition to the 'matched' counter, to allow for more actions than just dropping the packet
- the qname map now uses the qname and qtype as key, instead of using only the qname, and the value contains the action and counter fields described above instead of having a counter and the qtype

The first, legacy format is still used because of the limitations of eBPF socket filter programs on older kernels, and the number of instructions in particular, that prevented us from using the qname and qtype as key. We will likely switch to the newer format by default once Linux distributions stop shipping these older kernels. XDP programs require newer kernel versions anyway and have thus fewer limitations.

XDP programs are more powerful than eBPF socket filtering ones as they are not limited to accepting or denying a packet, but can immediately craft and send an answer. They are also executed a bit earlier in the kernel networking path so can provide better performance.

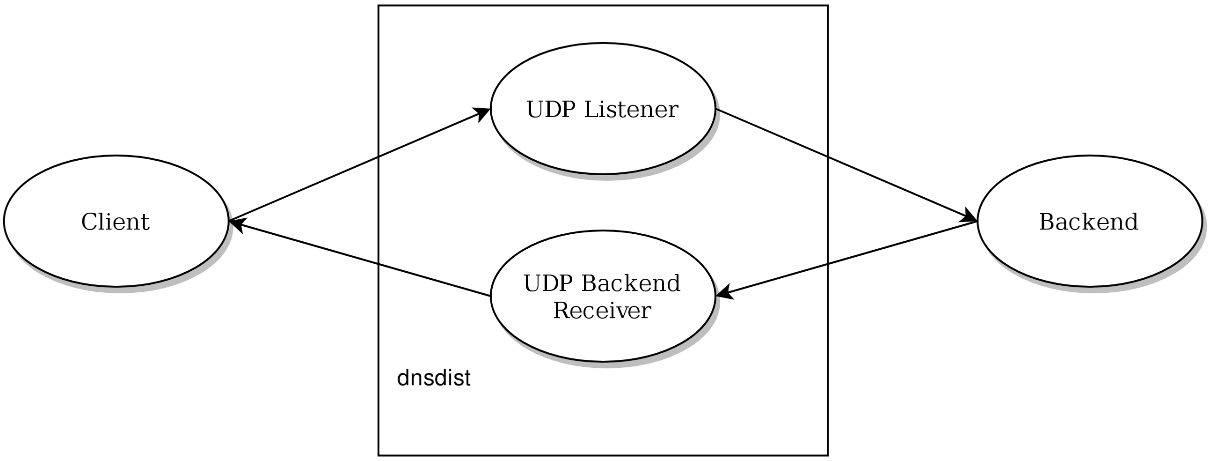
A sample program using the maps populated by dnsdist in an external XDP program can be found in the contrib/ directory of our git repository. That program supports answering with a TC=1 response instead of simply dropping the packet.

18.8 Performance Tuning

First, a few words about **dnsdist** architecture:

- Each local bind has its own thread listening for incoming UDP queries
- and its own thread listening for incoming TCP connections, dispatching them right away to a pool of TCP worker threads
- Each backend has its own thread listening for UDP responses, including the ones triggered by DoH queries, if any
- A maintenance thread calls the maintenance() Lua function every second if any, and is responsible for cleaning the cache
- A health check thread checks the backends availability
- A control thread handles console connections, plus one thread per connection
- A carbon thread exports statistics to carbon servers if needed
- One or more webserver threads handle queries to the internal webserver, plus one thread per HTTP connection
- A SNMP thread handles SNMP operations, when enabled.

18.8.1 UDP and incoming DNS over HTTPS



dnsdist design choices mean that the processing of UDP and DNS over HTTPS queries is done by only one thread per local bind (per *addLocal()*, addDNSCryptLocal() and *addDOHLocal()* directive).

This is great to keep lock contention to a low level, but might not be optimal for setups using a lot of processing power, caused for example by a large number of complicated rules. To be able to use more CPU cores for UDP queries processing, it is possible to use the reusePort parameter of the *addLocal()* and *setLocal()* directives to be able to add several identical local binds to dnsdist:

```
addLocal("192.0.2.1:53", {reusePort=true})
addLocal("192.0.2.1:53", {reusePort=true})
addLocal("192.0.2.1:53", {reusePort=true})
addLocal("192.0.2.1:53", {reusePort=true})
```

dnsdist will then add four identical local binds as if they were different IPs or ports, start four threads to handle incoming queries and let the kernel load balance those randomly to the threads, thus using four CPU cores for rules processing. Note that this require SO_REUSEPORT support in the underlying operating system (added for example in Linux 3.9). Please also be aware that doing so will increase lock contention and might not therefore scale linearly, as discussed below.

Another possibility is to use the reuseport option to run several dnsdist processes in parallel on the same host, thus avoiding the lock contention issue at the cost of having to deal with the fact that the different processes will not share informations, like statistics or DDoS offenders.

The UDP threads handling the responses from the backends do not use a lot of CPU, but if needed it is also possible to add the same backend several times to the dnsdist configuration to distribute the load over several responder threads:

```
newServer({address="192.0.2.127:53", name="Backend1"})
newServer({address="192.0.2.127:53", name="Backend2"})
newServer({address="192.0.2.127:53", name="Backend3"})
newServer({address="192.0.2.127:53", name="Backend4"})
```

When dispatching UDP queries to backend servers, dnsdist keeps track of at most **n** outstanding queries for each backend. This number **n** can be tuned by the *setMaxUDPOutstanding()* directive, defaulting to 65535 which is the maximum value.

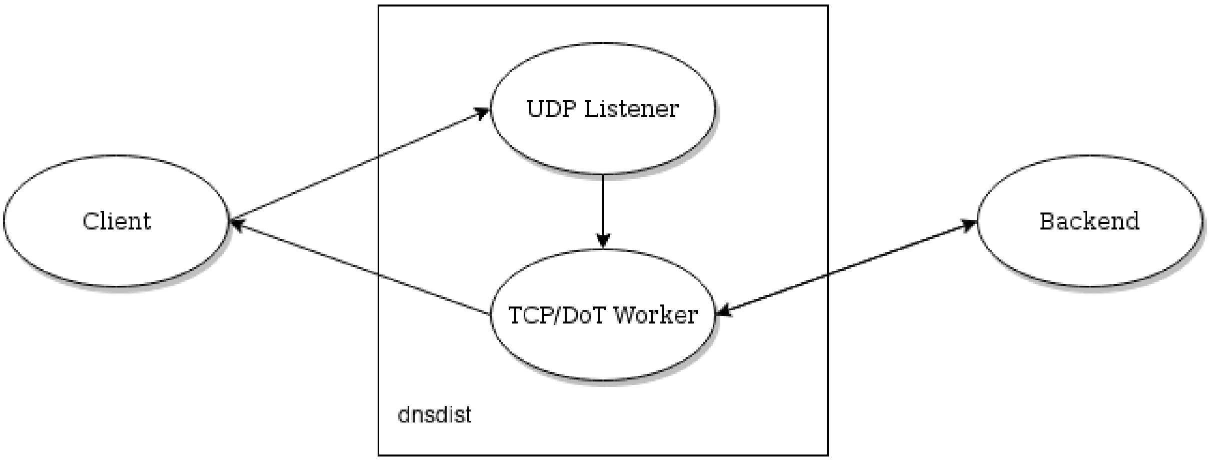
Changed in version 1.4.0: The default was 10240 before 1.4.0

Large installations running dnsdist before 1.4.0 are advised to increase the default value at the cost of a slightly increased memory usage.

Looking at `udp-in-errors` in *dumpStats()* will reveal whether the system is dropping UDP datagrams because dnsdist does not pick them up fast enough. In that case it might be good to add more *addLocal()* directives. In the same way, if the number of `Drops` in *showServers()* increase fast enough, it might mean that the backend is overloaded but also that the UDP received thread is. In that case adding more *newServer()*

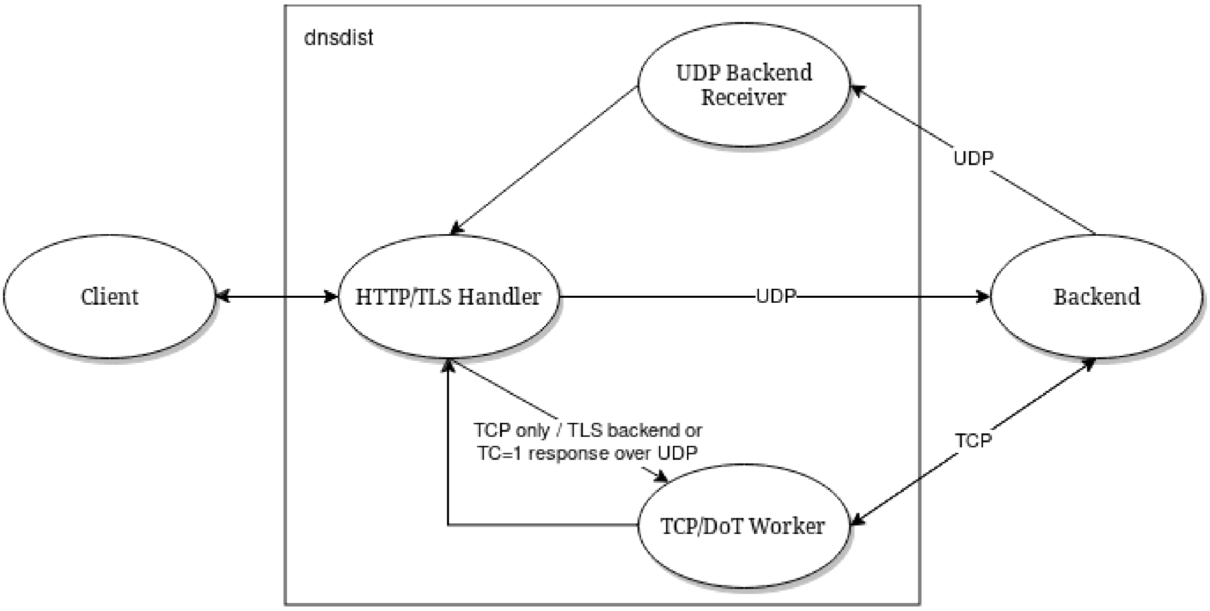
Using a single connected UDP socket to contact a backend, and thus a single (source address, source port, destination address, destination port) tuple, might not play well with some load-balancing mechanisms present in front of the backend. Linux's `reuseport`, for example, does not balance the incoming datagrams to several threads in that case. That can be worked around by using the `sockets` option of the *newServer()* directive to open several sockets instead of one. You may want to set that number to a value somewhat higher than the number of worker threads configured in the backend. dnsdist will then select a socket using round-robin to forward a query to the backend, and use event multiplexing on the receiving side.

Note that, since 1.7, dnsdist supports marking a backend as "TCP only", as well as enabling DNS over TLS communication between dnsdist and that backend. That leads to a different model where UDP queries are instead passed to a TCP worker:



For DNS over HTTPS, every *addDOHLocal()*/*addDOH3Local()* directive adds a new thread dealing with incoming connections, so it might be useful to add more than one directive, as indicated above.

When dealing with a large traffic load, it might happen that the internal pipe used to pass queries between the threads handling the incoming connections and the one getting a response from the backend become full too quickly, degrading performance and causing timeouts. This can be prevented by increasing the size of the internal pipe buffer, via the *internalPipeBufferSize* option of *addDOHLocal()*. Setting a value of *1048576* is known to yield good results on Linux.



18.8.2 *AF_XDP* / *XSK*

On recent versions of Linux (>= *4.18*), DNSDist supports receiving UDP datagrams directly from the kernel, bypassing the usual network stack, via *AF_XDP*/*XSK*. This yields much better performance but comes with some limitations. Please see *AF_XDP / XSK* for more information.

18.8.3 UDP buffer sizes

The operating system usually maintains buffers of incoming and outgoing datagrams for UDP sockets, to deal with short spikes where packets are received or emitted faster than the network layer can process them. On medium to large setups, it is usually useful to increase these buffers to deal with large spikes. This can be done via the `setUDPSocketBufferSizes()`.

18.8.4 Outgoing DoH

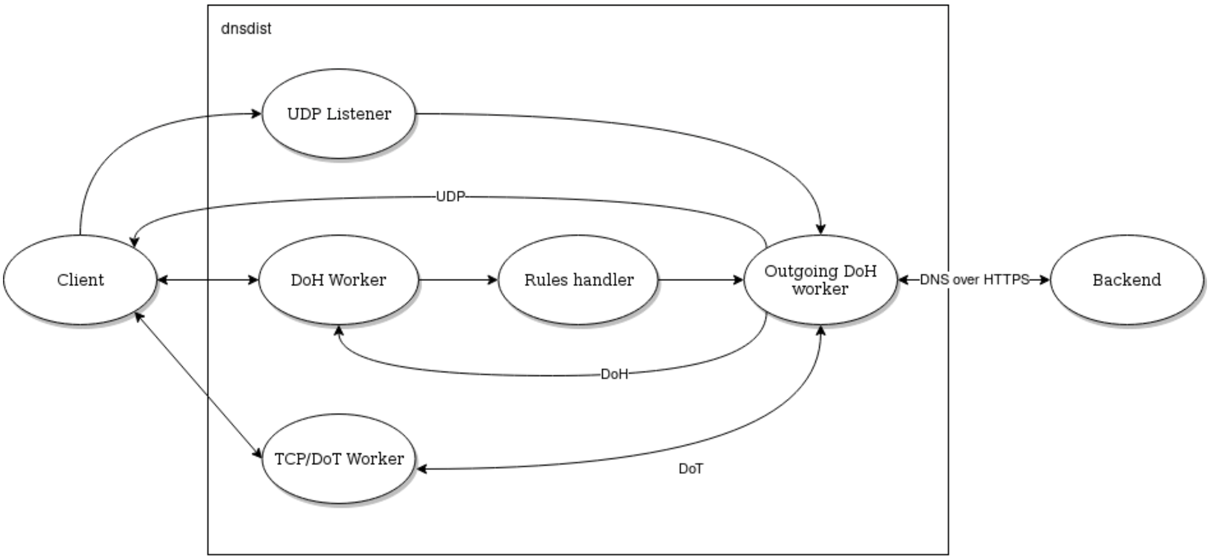
Starting with 1.7.0, dnsdist supports communicating with the backend using DNS over HTTPS. The incoming queries, after the processing of rules if any, are passed to one of the DoH workers over a pipe. The DoH worker handles the communication with the backend, retrieves the response, and either responds directly to the client (queries coming over UDP) or pass it back over a pipe to the initial thread (queries coming over TCP, DoT or DoH). The number of outgoing DoH worker threads can be configured using `setOutgoingDoHWorkerThreads()`.

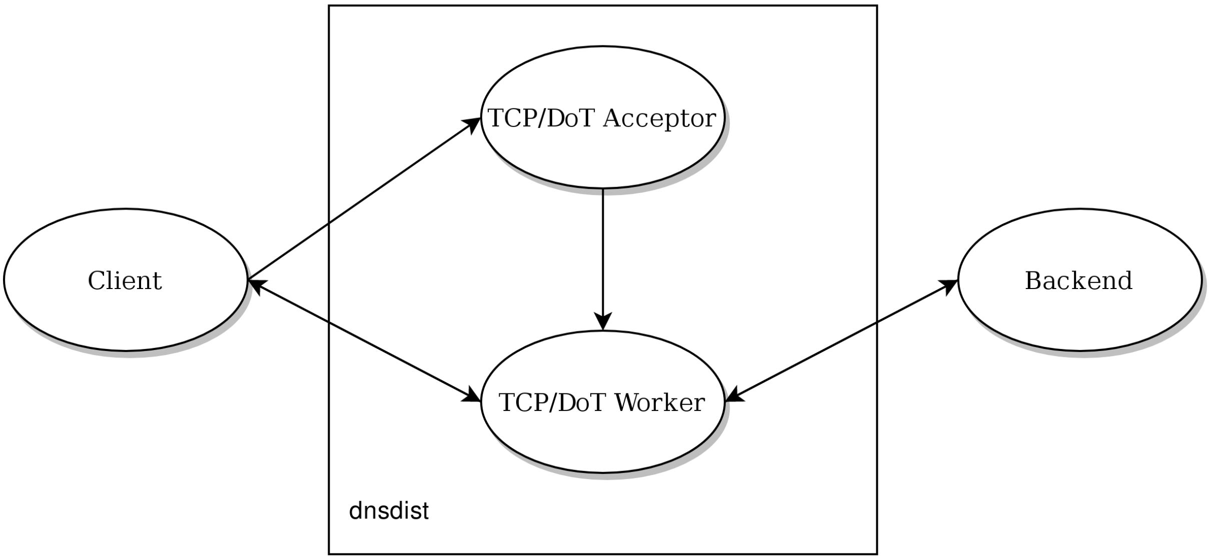
18.8.5 TCP and DNS over TLS

Before 1.4.0, a TCP thread could only handle a single incoming connection at a time. Starting with 1.4.0 the handling of TCP connections is now event-based, so a single TCP worker can handle a large number of TCP incoming connections simultaneously. Note that before 1.6.0 the TCP worker threads were created at runtime, adding a new thread when the existing ones seemed to struggle with the load, until the maximum number of threads had been reached. Starting with 1.6.0 the configured number of worker threads are immediately created at startup.

The maximum number of threads in the TCP / DNS over TLS pool is controlled by the `setMaxTCPClientThreads()` directive, and defaults to 10. This number can be increased to handle a large number of simultaneous TCP / DNS over TLS connections.

If all the TCP threads are busy, new TCP connections are queued while they wait to be picked up. The maximum number of queued connections can be configured with `setMaxTCPQueuedConnections()` and defaults to


dnsdist
UDP Listener
UDP
Client
DoH Worker
Rules handler
Outgoing DoH worker
DNS over HTTPS
Backend
DoH
TCP/DoT Worker
DoT


TCP/DoT Acceptor
Client
Backend
TCP/DoT Worker
dnsdist

1000 (10000 on Linux since 1.6.0). Note that the size of the internal pipe used to distribute queries might need to be increased as well, using `setTCPInternalPipeBufferSize()`. Any value larger than 0 will cause new connections to be dropped if there are already too many queued.

By default, every TCP worker thread has its own queue, and the incoming TCP connections are dispatched to TCP workers on a round-robin basis. This might cause issues if some connections are taking a very long time, since incoming ones will be waiting until the TCP worker they have been assigned to has finished handling its current query, while other TCP workers might be available.

The experimental `setTCPUseSinglePipe()` directive can be used so that all the incoming TCP connections are put into a single queue and handled by the first TCP worker available. This used to be useful before 1.4.0 because a single connection could block a TCP worker, but the "one pipe per TCP worker" is preferable now that workers can handle multiple connections to prevent waking up all idle workers when a new connection arrives. This option will be removed in 1.7.0.

One of the first starting point when investigating TCP or DNS over TLS issues is to look at the `showTCPStats()` command. It provides a lot of metrics about the current and passed connections, and why they were closed.

If the number of queued connections ("Queued" in `showTCPStats()`) reaches the maximum number of queued connections ("Max Queued" in `showTCPStats()`) then there is clearly a problem with TCP workers not picking up new connections quickly enough. It might be a good idea to increase the number of TCP workers.

A different possibility is that there is not enough threads accepting new connections and distributing them to worker threads. Looking at whether the `listenOverflows` metric in `dumpStats()` increase over time will tell if we are losing TCP connections because the queue is full. In that case, since a single `addLocal()` or `addTLSLocal()` directive results in only one acceptor thread, it might useful to add more of these.

For incoming and outgoing DNS over TLS support, the choice of the TLS provider (OpenSSL and GnuTLS are both supported) might yield very different results depending on the exact architecture.

Incoming DNS over TLS (since 1.8.0) and incoming DNS over HTTPS (since 1.9.0) might also benefit from experimental support for TLS acceleration engines, like Intel QAT. See `loadTLSEngine()`, and the *tlsAsyncMode* parameter of `addTLSLocal()` and `addDOHLocal()` for more information.

Incoming and outgoing DNS over TLS, outgoing DNS over HTTPS, as well as incoming DNS over HTTPS with the `nghttp2` library (since 1.9.0), might benefit from experimental support kernel-accelerated TLS on Linux, when supported by the kernel and OpenSSL. See the *ktls* options on `addTLSLocal()`, `addDOHLocal()` and `newServer()` for more information. Kernel support for kTLS might be verified by looking at the counters in `/proc/net/tls_stat`. Note that:

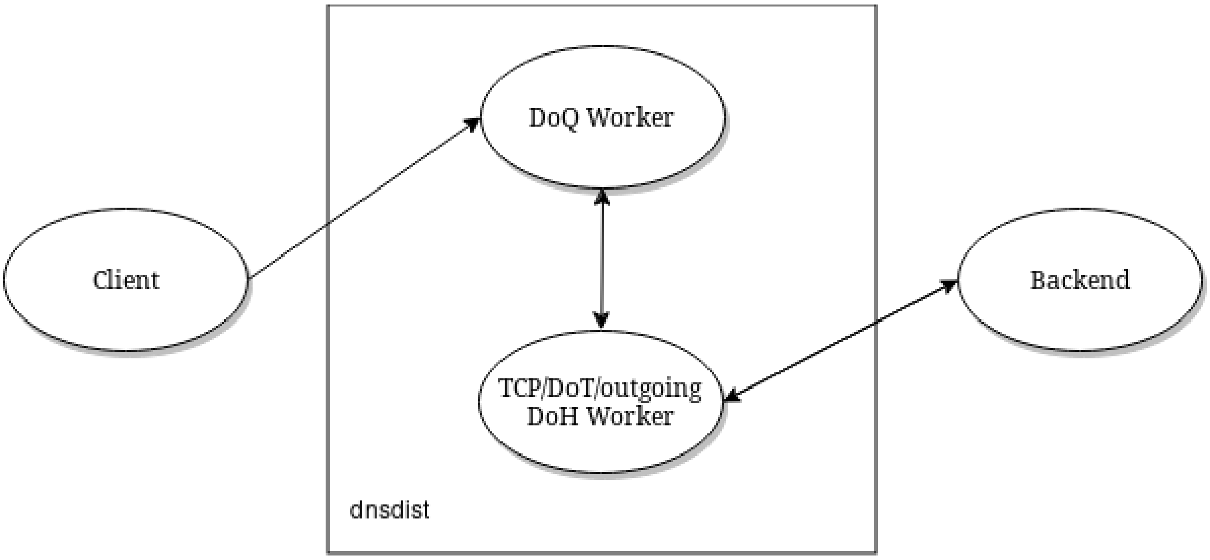
- supported ciphers depend on the exact kernel version used. `TLS_AES_128_GCM_SHA256` might be a good option for testing purpose since it was supported pretty early
- as of OpenSSL 3.0.7, kTLS can only be used for sending TLS 1.3 packets, not receiving them. Both sending and receiving packets should be working for TLS 1.2.

18.8.6 TLS performance

For DNS over HTTPS and DNS over TLS, in addition to the advice above we suggest reading the *TLS Sessions Management* page to learn how to improve TLS session resumption ratio, which has a huge impact on CPU usage and latency.

18.8.7 DNS over QUIC

For DNS over QUIC, every `addDOQLocal()` directive adds a new thread dealing with incoming datagrams, so it might be useful to add more than one directive.



18.8.8 Rules and Lua

Most of the query processing is done in C++ for maximum performance, but some operations are executed in Lua for maximum flexibility:

- Rules added by `LuaAction()`, `LuaResponseAction()`, `LuaFFIAction()` or `LuaFFIResponseAction()`
- Server selection policies defined via `setServerPolicyLua()`, `setServerPolicyLuaFFI()`, `setServerPolicyLuaFFIPerThread()` or `newServerPolicy()`

While Lua is fast, its use should be restricted to the strict necessary in order to achieve maximum performance, it might be worth considering using LuaJIT instead of Lua. When Lua inspection is needed, the best course of action is to restrict the queries sent to Lua inspection by using `addLuaAction()` with a selector.

Type	Performance	Locking
C++ rule	fast	none
Lua rue	slow	global Lua lock
Lua FFI rule	fast	global Lua lock
Lua per-thread FFI rule	fast	none
C++ LB policy	fast	none
Lua LB policy	slow	global Lua lock
Lua FFI LB policy	fast	global Lua lock
Lua per-thread FFI LB policy	fast	none

18.8.9 Lock contention and sharding

Adding more threads makes it possible to use more CPU cores to deal with the load, but at the cost of possibly increasing lock contention between threads. This is especially true for Lua-intensive setups, because Lua processing in dnsdist is serialized by a unique lock for all threads, as seen above.

For other components, like the packet cache and the in-memory ring buffers, it is possible to reduce the amount of contention by using sharding. Sharding divides the memory into several pieces, every one of these having its own separate lock, reducing the amount of times two threads or more will need to access the same data.

Sharding was disabled by default before 1.6.0 and could be enabled via the *numberOfShards* option to `newPacketCache()` and `setRingBuffersSize()`. It might still make sense to increment the number of shards when dealing with a lot of threads.

18.8.10 Memory usage

The main sources of memory usage in DNSDist are:

- packet caches, when enabled
- the number of outstanding UDP queries per backend, configured with *setMaxUDPOutstanding()* (see above)
- the number of entries in the ring-buffers, configured with *setRingBuffersSize()*
- the number of short-lived dynamic block entries
- the number of user-defined rules and actions
- the number of TCP, DoT and DoH connections

Memory usage per connection for connected protocols:

Protocol	Memory usage per connection
TCP	6 kB
DoT (GnuTLS)	16 kB
DoT (OpenSSL)	52 kB
DoT (OpenSSL w/ releaseBuffers)	19 kB
DoH (http)	2 kB
DoH	48 kB
DoH (w/ releaseBuffers)	15 kB

18.8.11 Firewall connection tracking

When dealing with a lot of queries per second, dnsdist puts a severe stress on any stateful (connection tracking) firewall, so much so that the firewall may fail.

Specifically, many Linux distributions run with a connection tracking firewall configured. For high load operation (thousands of queries/second), it is advised to either turn off `iptables` and `nftables` completely, or use the `NOTRACK` feature to make sure client DNS traffic bypasses the connection tracking.

18.8.12 Network interface receive queues

Most high-speed (>= 10 Gbps) network interfaces support multiple queues to offer better performance, using hashing to dispatch incoming packets into a specific queue.

Unfortunately the default hashing algorithm is very often considering the source and destination addresses only, which might be an issue when dnsdist is placed behind a frontend, for example.

On Linux it is possible to inspect the current network flow hashing policy via `ethtool`:

```
$ sudo ethtool -n enp1s0 rx-flow-hash udp4
UDP over IPV4 flows use these fields for computing Hash flow key:
IP SA
IP DA
```

In this example only the source (`IP SA`) and destination (`IP DA`) addresses are indeed used, meaning that all packets coming from the same source address to the same destination address will end up in the same receive queue, which is not optimal. To take the source and destination ports into account as well:

```
$ sudo ethtool -N enp1s0 rx-flow-hash udp4 sdfn
$
```

18.9 SNMP support

dnsdist supports exporting statistics and sending traps over SNMP when compiled with `Net SNMP` support, acting as an `AgentX` subagent. SNMP support is enabled via the *snmpAgent()* directive.

By default, the only traps sent when Traps are enabled, are backend status change notifications. But custom traps can also be sent:

- from Lua, with *sendCustomTrap()* and *DNSQuestion:sendTrap()*
- For selected queries and responses, using *SNMPTrapAction()* and *SNMPTrapResponseAction()*

`Net SNMP snmpd` doesn't accept subagent connections by default, so to use the SNMP features of **dnsdist** the following line should be added to the `snmpd.conf` configuration file:

```
master agentx
```

In addition to that, the permissions on the resulting socket might need to be adjusted so that the `dnsdist` user can write to it. This can be done with the following lines in `snmpd.conf` (assuming *dnsdist* is running as *dnsdist:dnsdist*):

```
agentxperms 0700 0700 dnsdist dnsdist
```

In order to allow the retrieval of statistics via SNMP, `snmpd`'s access control has to configured. A very simple SNMPv2c setup only needs the configuration of a read-only community in `snmpd.conf`:

```
rocommunity dnsdist42
```

`snmpd` also supports more secure SNMPv3 setup, using for example the `createUser` and `rouser` directives:

```
createUser myuser SHA "my auth key" AES "my enc key"
rouser myuser
```

`snmpd` can be instructed to send SNMPv2 traps to a remote SNMP trap receiver by adding the following directive to the `snmpd.conf` configuration file:

```
trap2sink 192.0.2.1
```

The description of **dnsdist**'s SNMP MIB is as follows:

```
-- -*- snmpv2 -*-
-- ----------------------------------------------------------------------
-- MIB file for dnsdist
-- ----------------------------------------------------------------------

DNSDIST-MIB DEFINITIONS ::= BEGIN

IMPORTS
    OBJECT-TYPE, MODULE-IDENTITY, enterprises,
    Counter64, Unsigned32, NOTIFICATION-TYPE
        FROM SNMPv2-SMI
    CounterBasedGauge64
        FROM HCNUM-TC
    Float64TC
        FROM FLOAT-TC-MIB
    OBJECT-GROUP, MODULE-COMPLIANCE, NOTIFICATION-GROUP
        FROM SNMPv2-CONF
    InetAddressType
        FROM INET-ADDRESS-MIB
    TEXTUAL-CONVENTION, DisplayString
        FROM SNMPv2-TC;

```

(continues on next page)

(continued from previous page)

```
dnsdist MODULE-IDENTITY
    LAST-UPDATED "201611080000Z"
    ORGANIZATION "PowerDNS BV"
    CONTACT-INFO "support@powerdns.com"
    DESCRIPTION
       "This MIB module describes information gathered through dnsdist."

    REVISION "201611080000Z"
    DESCRIPTION "Initial revision."

    ::= { powerdns 3 }

powerdns                OBJECT IDENTIFIER ::= { enterprises 43315 }

stats OBJECT IDENTIFIER ::= { dnsdist 1 }

queries OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of queries received"
    ::= { stats 1 }

responses OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of responses received"
    ::= { stats 2 }

servfailResponses OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of servfail responses received"
    ::= { stats 3 }

aclDrops OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of queries dropped because of the ACL"
    ::= { stats 4 }

-- stats 5 was a BlockFilter Counter, removed in 1.2.0

ruleDrop OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of queries dropped because of a rule"
    ::= { stats 6 }

ruleNXDomain OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
```

(continues on next page)

(continued from previous page)

```
    STATUS current
    DESCRIPTION
        "Number of NXDomain responses returned because of a rule"
    ::= { stats 7 }

ruleRefused OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of Refused responses returned because of a rule"
    ::= { stats 8 }

selfAnswered OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of self-answered responses"
    ::= { stats 9 }

downstreamTimeouts OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of downstream timeouts"
    ::= { stats 10 }

downstreamSendErrors OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of downstream send errors"
    ::= { stats 11 }

truncFailures OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of errors while truncating a response"
    ::= { stats 12 }

noPolicy OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of queries dropped because no server was available"
    ::= { stats 13 }

latency01 OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of UDP queries answered in less than 1 ms"
    ::= { stats 14 }
```

(continues on next page)

(continued from previous page)

```
latency110 OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of UDP queries answered in 1-10 ms"
    ::= { stats 15 }

latency1050 OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of UDP queries answered in 10-50 ms"
    ::= { stats 16 }

latency50100 OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of UDP queries answered in 50-100 ms"
    ::= { stats 17 }

latency1001000 OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of UDP queries answered in 100-1000 ms"
    ::= { stats 18 }

latencySlow OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of UDP queries answered in more than 1s"
    ::= { stats 19 }

latencyAVG100 OBJECT-TYPE
    SYNTAX Float64TC
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Average latency over the last 100 queries"
    ::= { stats 20 }

latencyAVG1000 OBJECT-TYPE
    SYNTAX Float64TC
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Average latency over the last 1000 queries"
    ::= { stats 21 }

latencyAVG10000 OBJECT-TYPE
    SYNTAX Float64TC
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
```

(continues on next page)

(continued from previous page)

```
        "Average latency over the last 10000 queries"
    ::= { stats 22 }

latencyAVG1000000 OBJECT-TYPE
    SYNTAX Float64TC
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Average latency over the last 1000000 queries"
    ::= { stats 23 }

uptime OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Uptime of the dnsdist process, in seconds"
    ::= { stats 24 }

realMemoryUsage OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Memory usage"
    ::= { stats 25 }

nonCompliantQueries OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of queries dropped as non-compliant"
    ::= { stats 26 }

nonCompliantResponses OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of responses dropped as non-compliant"
    ::= { stats 27 }

rdQueries OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of queries with the RD flag set"
    ::= { stats 28 }

emptyQueries OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of empty queries received"
    ::= { stats 29 }

cacheHits OBJECT-TYPE
    SYNTAX Counter64
```

(continues on next page)

(continued from previous page)

```
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of cache hits"
    ::= { stats 30 }

cacheMisses OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of cache misses"
    ::= { stats 31 }

cpuUserMSec OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "CPU Usage (user)"
    ::= { stats 32 }

cpuSysMSec OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "CPU Usage (sys)"
    ::= { stats 33 }

fdUsage OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of file descriptors"
    ::= { stats 34 }

dynBlocked OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of queries dropped because of a dynamic block"
    ::= { stats 35 }

dynBlockNMGSize OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Dynamic blocks (NMG) size"
    ::= { stats 36 }

ruleServFail OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of ServFail responses returned because of a rule"
    ::= { stats 37 }
```

(continues on next page)

(continued from previous page)

```
securityStatus OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Security status of this software. 0=unknown, 1=OK, 2=upgrade recommended,␣
↪3=upgrade mandatory"
    ::= { stats 38 }

specialMemoryUsage OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Memory usage (more precise but expensive to retrieve)"
    ::= { stats 39 }

ruleTruncated OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of Truncated responses returned because of a rule"
    ::= { stats 40 }

backendStatTable OBJECT-TYPE
    SYNTAX SEQUENCE OF BackendStatEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION "Statistics for backends"
    ::= { dnsdist 2 }

backendStatEntry OBJECT-TYPE
    SYNTAX BackendStatEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION "Statistics for one backend"
    INDEX { backendId }
    ::= { backendStatTable 1 }

BackendStatEntry ::= SEQUENCE {
    backendId           Unsigned32,
    backendName         DisplayString,
    backendLatency      CounterBasedGauge64,
    backendWeight       CounterBasedGauge64,
    backendOutstanding  CounterBasedGauge64,
    backendQPSLimit     CounterBasedGauge64,
    backendReused       Counter64,
    backendState        DisplayString,
    backendAddress      OCTET STRING,
    backendPools        DisplayString,
    backendQPS          CounterBasedGauge64,
    backendQueries      Counter64,
    backendOrder        CounterBasedGauge64
}

backendId OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS not-accessible
    STATUS current
```

(continues on next page)

(continued from previous page)

```
    DESCRIPTION
        "Backend ID"
    ::= { backendStatEntry 1 }

backendName OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend name"
    ::= { backendStatEntry 2 }

backendLatency OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend latency"
    ::= { backendStatEntry 3 }

backendWeight OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend weight"
    ::= { backendStatEntry 4 }

backendOutstanding OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend outstanding queries"
    ::= { backendStatEntry 5 }

backendQPSLimit OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend QPS limit"
    ::= { backendStatEntry 6 }

backendReused OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend reused slots"
    ::= { backendStatEntry 7 }

backendState OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend state"
    ::= { backendStatEntry 8 }

backendAddress OBJECT-TYPE
```

(continues on next page)

(continued from previous page)

```
    SYNTAX OCTET STRING (SIZE (2..24))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend address"
    ::= { backendStatEntry 9 }

backendPools OBJECT-TYPE
    SYNTAX DisplayString
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "List of pools this backend belongs to"
    ::= { backendStatEntry 10 }

backendQPS OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend QPS"
    ::= { backendStatEntry 11 }

backendQueries OBJECT-TYPE
    SYNTAX Counter64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Number of queries sent to this backend"
    ::= { backendStatEntry 12 }

backendOrder OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Backend order"
    ::= { backendStatEntry 13 }

---
--- Textual Conventions
---

SocketProtocolType ::= TEXTUAL-CONVENTION
    STATUS      current
    DESCRIPTION
        "A value that represents a type of socket protocol."
    SYNTAX       INTEGER {
                     unknown(0),
                     udp(1),
                     tcp(2)
                 }

DNSQueryType ::= TEXTUAL-CONVENTION
    STATUS      current
    DESCRIPTION
        "A value that represents a type of DNS query (question or response)."
    SYNTAX       INTEGER {
                     unknown(0),
                     question(1),
                     response(2)
```

(continues on next page)

(continued from previous page)

```
                  }

---
--- Traps / Notifications
---

trap OBJECT IDENTIFIER ::= { dnsdist 10 }
traps OBJECT IDENTIFIER ::= { trap 0 } --- reverse-mappable
trapObjects OBJECT IDENTIFIER ::= { dnsdist 11 }

socketFamily OBJECT-TYPE
    SYNTAX InetAddressType
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Socket family type"
    ::= { trapObjects 1 }

socketProtocol OBJECT-TYPE
    SYNTAX SocketProtocolType
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Socket protocol type"
    ::= { trapObjects 2 }

fromAddress OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (2..24))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Requestor address"
    ::= { trapObjects 3 }

toAddress OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (2..24))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Responder address"
    ::= { trapObjects 4 }

queryType OBJECT-TYPE
    SYNTAX DNSQueryType
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Query / Response"
    ::= { trapObjects 5 }

querySize OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Size in bytes"
    ::= { trapObjects 6 }

queryID OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-only
```

(continues on next page)

(continued from previous page)

```
    STATUS current
    DESCRIPTION
        "DNS query ID"
    ::= { trapObjects 7 }

qName OBJECT-TYPE
    SYNTAX OCTET STRING (SIZE (0..255))
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "DNS qname"
    ::= { trapObjects 8 }

qClass OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "DNS query class"
    ::= { trapObjects 9 }

qType OBJECT-TYPE
    SYNTAX Unsigned32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "DNS query type"
    ::= { trapObjects 10 }

trapReason OBJECT-TYPE
    SYNTAX OCTET STRING
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Reason for this trap"
    ::= { trapObjects 11 }

--- { trapObjects 5000 } up to and including { trapObjects 5999 } are reserved for␣
↪local, product-specific extensions to the dnsdist MIB

backendStatusChangeTrap NOTIFICATION-TYPE
    OBJECTS {
        backendName,
        backendAddress,
        backendState
    }
    STATUS current
    DESCRIPTION "Backend status changed"
    ::= { traps 1 }

actionTrap NOTIFICATION-TYPE
    OBJECTS {
        socketFamily,
        socketProtocol,
        fromAddress,
        toAddress,
        queryType,
        querySize,
        queryID,
        qName,
        qClass,
```

(continues on next page)

(continued from previous page)

```
        qType,
        trapReason
    }
    STATUS current
    DESCRIPTION "Trap sent by SNMPTrapAction"
    ::= { traps 2 }

customTrap NOTIFICATION-TYPE
    OBJECTS {
        trapReason
    }
    STATUS current
    DESCRIPTION "Trap sent by sendCustomTrap"
    ::= { traps 3 }

--- { traps 5000 } up to and including { traps 5999 } are reserved for local,␣
↪product-specific extensions to the dnsdist MIB

---
--- Conformance
---

dnsdistConformance OBJECT IDENTIFIER ::= { dnsdist 100 }

dnsdistCompliances MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION "dnsdist compliance statement"
    MODULE
    MANDATORY-GROUPS {
        dnsdistGroup,
        dnsdistTrapsGroup
    }
    ::= { dnsdistConformance 1 }

dnsdistGroup OBJECT-GROUP
    OBJECTS {
        queries,
        responses,
        servfailResponses,
        aclDrops,
        ruleDrop,
        ruleNXDomain,
        ruleRefused,
        ruleServFail,
        ruleTruncated,
        selfAnswered,
        downstreamTimeouts,
        downstreamSendErrors,
        truncFailures,
        noPolicy,
        latency01,
        latency110,
        latency1050,
        latency50100,
        latency1001000,
        latencySlow,
        latencyAVG100,
        latencyAVG1000,
        latencyAVG10000,
        latencyAVG1000000,
        uptime,
```

(continues on next page)

(continued from previous page)

```
        realMemoryUsage,
        specialMemoryUsage,
        nonCompliantQueries,
        nonCompliantResponses,
        rdQueries,
        emptyQueries,
        cacheHits,
        cacheMisses,
        cpuUserMSec,
        cpuSysMSec,
        fdUsage,
        dynBlocked,
        dynBlockNMGSize,
        securityStatus,
        backendName,
        backendLatency,
        backendWeight,
        backendOutstanding,
        backendQPSLimit,
        backendReused,
        backendState,
        backendAddress,
        backendPools,
        backendQPS,
        backendQueries,
        backendOrder,
        socketFamily,
        socketProtocol,
        fromAddress,
        toAddress,
        queryType,
        querySize,
        queryID,
        qName,
        qClass,
        qType,
        trapReason
    }
    STATUS current
    DESCRIPTION "Objects conformance group for dnsdist"
    ::= { dnsdistConformance 2 }

dnsdistTrapsGroup NOTIFICATION-GROUP
    NOTIFICATIONS {
        actionTrap,
        customTrap,
        backendStatusChangeTrap
    }
    STATUS current
    DESCRIPTION "Traps conformance group for dnsdist"
    ::= { dnsdistConformance 3 }

END
```

18.10 AXFR, IXFR and NOTIFY

18.10.1 In front of primaries

When **dnsdist** is deployed in front of a primary authoritative server, it might receive AXFR or IXFR queries destined to this primary. There are two issues that can arise in this kind of setup:

- If the primary is part of a pool of servers, the first *SOA* query can be directed by **dnsdist** to a different server than the following AXFR/IXFR one, which might fail if the servers are not perfectly synchronised.
- If the primary only allows AXFR/IXFR based on the source address of the requestor, it might be confused by the fact that the source address will be the one from the **dnsdist** server.

The first issue can be solved by routing SOA, AXFR and IXFR requests explicitly to the primary:

```
newServer({address="192.168.1.2", name="primary", pool={"primary", "otherpool"}})
addAction(OrRule({QTypeRule(DNSQType.SOA), QTypeRule(DNSQType.AXFR),␣
↪QTypeRule(DNSQType.IXFR)}), PoolAction("primary"))
```

The second one might require allowing AXFR/IXFR from the **dnsdist** source address and moving the source address check to **dnsdist**'s side:

```
addAction(AndRule({OrRule({QTypeRule(DNSQType.AXFR), QTypeRule(DNSQType.IXFR)}),␣
↪NotRule(makeRule("192.168.1.0/24"))}), RCodeAction(DNSRCode.REFUSED))
```

Changed in version 1.4.0: Before 1.4.0, the QTypes were in the `dnsdist` namespace. Use `dnsdist.AXFR` and `dnsdist.IXFR` in these versions. Before 1.4.0, the RCodes were in the `dnsdist` namespace. Use `dnsdist.REFUSED` in these versions.

A different way would be to configure dnsdist to pass the source IP of the client to the backend. The different options to do that are described in *Passing the source address to the backend*.

Warning: Be wary of dnsdist caching the responses to AXFR and IXFR queries and sending these to the wrong clients. This is mitigated by default when the source IP of the client is passed using EDNS Client Subnet, but not when the proxy protocol is used, so disabling caching for these kinds of queries is advised:

```
-- this rule will not stop the processing, but disable caching for AXFR and IXFR␣
↪responses
addAction(OrRule({QTypeRule(DNSQType.AXFR), QTypeRule(DNSQType.IXFR)}),␣
↪SetSkipCacheAction())
-- this rule will route SOA, AXFR and IXFR queries to a specific pool of servers
addAction(OrRule({QTypeRule(DNSQType.SOA), QTypeRule(DNSQType.AXFR),␣
↪QTypeRule(DNSQType.IXFR)}), PoolAction("primary"))
```

Changed in version 1.8.0: Since 1.8.0, dnsdist will no longer cache responses to AXFR and IXFR queries.

18.10.2 In front of secondaries

When **dnsdist** is deployed in front of secondaries, however, an issue might arise with NOTIFY queries, because the secondary will receive a notification coming from the **dnsdist** address, and not the primary's one. One way to fix this issue is to allow NOTIFY from the **dnsdist** address on the secondary side (for example with PowerDNS's *trusted-notification-proxy*) and move the address check to **dnsdist**'s side:

```
addAction(AndRule({OpcodeRule(DNSOpcode.Notify), NotRule(makeRule("192.168.1.0/24
↪"))}), RCodeAction(DNSRCode.REFUSED))
```

Changed in version 1.4.0: Before 1.4.0, the RCodes were in the `dnsdist` namespace. Use `dnsdist.REFUSED` in these versions.

Warning: Be wary of dnsdist caching the responses to NOTIFY queries and sending these to the wrong clients. This is mitigated by default when the source IP of the client is passed using EDNS Client Subnet, but not when the proxy protocol is used, so disabling caching for these kinds of queries is advised:

```
-- this rule will disable the caching of responses for NOTIFY queries
addAction(OpcodeRule(DNSOpcode.Notify), SetSkipCacheAction())
```

18.11 Running multiple instances

Sometimes, it can be advantageous to run multiple instances of **dnsdist**. Usecases can be:

- Multiple inbound IP addresses with different rulesets
- Taking advantage of more processes, using SO_REUSEPORT

dnsdist supports loading a different configuration file with the `--config` command line switch.

By default, `SYSCONFDIR/dnsdist.conf` is loaded. `SYSCONFDIR` is usually `/etc` or `/etc/dnsdist`.

18.11.1 Using systemd

On systems with systemd, instance services can be used. To create a dnsdist service named `foo`, create a `dnsdist-foo.conf` in `SYSCONFDIR`, then run `systemctl enable dnsdist@foo.service` and `systemctl start dnsdist@foo.service`.

18.12 Out-of-order

As of 1.6.0, dnsdist supports accepting and processing queries out-of-order as long as the `maxInFlight` parameter has been set on the frontend, via *addLocal()* and/or *addTLSLocal()*. Note that it is always enabled on DoH frontends. As many as `maxInFlight` queries will then be read from a TCP connection, processed and forwarded to a backend simultaneously. If there is more queries pending, they will be processed once a response has been sent for one of the already processed queries.

Backends are assumed not to support out-of-order by default, so only one query at a time will be sent over a TCP connection to a backend, meaning that up to `maxInFlight` connections to a backend might be needed to be able to process all accepted queries. Setting `maxInFlight` to a value greater than zero on *newServer()* changes that, and up to `maxInFlight` queries can be sent to a backend simultaneously over the same TCP connection. This of course requires the backend to actually process incoming queries out-of-order, otherwise the latency will be considerably increased, leading to timeouts and degraded service.

As of 1.6.0, only queries from the same incoming client connection will be sent to a server over a single outgoing TCP connections. This will likely change in 1.7.0, once we have had time to check that it has no adverse effects.

Backends for which Proxy Protocol support has been enabled will never be able to reuse the same outgoing TCP connections for different clients, given that the payload indicating the source IP of the client, as seen by dnsdist, is sent once at the beginning of the TCP connection. For the same reason, it might not even be possible to reuse a TCP connection for the same client if any Type-Length-Value data has been sent over that connection.

18.13 OCSP Stapling

dnsdist supports OCSP stapling for DNS over HTTPS and DNS over TLS since 1.4.0-rc1. OCSP, Online Certificate Status Protocol (**RFC 6960**) is a protocol allowing a client to check the expiration status of a certificate from the certification authority (CA) that delivered it. Since the requirement for the client to first retrieve the certificate then do additional steps to gather an OCSP response is not very efficient, and also discloses to the CA which certificate is validated, a mechanism has been designed to allow the server to retrieve the OCSP response from the

CA and provide it to the client during the TLS exchange. This mechanism is named the TLS Certificate Status Request extension (**RFC 6066**), also known as OCSP stapling.

While OCSP stapling is a net win for the client, it means that the server needs to retrieve the OCSP response itself and update it at regular interval, since the OCSP response tends to be short-lived by design.

dnsdist, as for example haproxy, only supports loading the OCSP response from a file, and has no embedded HTTP client to retrieve the OCSP response and refresh it, leaving it to the administrator to regularly retrieve the OCSP response and feed it to dnsdist.

18.13.1 Local PKI

When a local PKI is used to issue the certificate, or for testing purposes, dnsdist provides the *`generateOCSPResponse()`* function to generate an OCSP response file for a certificate, using the certificate and private key of the certification authority that signed that certificate:

```
generateOCSPResponse(pathToServerCertificate, pathToCACertificate,␣
↪pathToCAPrivateKey, outputFile, numberOfDaysOfValidity,␣
↪numberOfMinutesOfValidity)
```

The resulting file can be directly used with the *`addDOHLocal()`* or the *`addTLSLocal()`* functions:

```
addDOHLocal("127.0.0.1:443", "/path/to/the/server/certificate", "/path/to/the/
↪server/private/key", { "/" }, { ocspResponses={"/path/to/generated/ocsp/response
↪"}})
addTLSLocal("127.0.0.1:853", "/path/to/the/server/certificate", "/path/to/the/
↪server/private/key", { ocspResponses={"/path/to/generated/ocsp/response"}})
```

After starting dnsdist, it is possible to update the OCSP response by connecting to the *console*, generating a new OCSP response and calling *`reloadAllCertificates()`* so that dnsdist reloads the certificates, keys and OCSP responses associated to the DNS over TLS and DNS over HTTPS contexts.

18.13.2 Certificate signed by an external authority

When the certificate has been signed by an external certification authority, the process is a bit more complicated because the OCSP needs to be retrieved from that CA, and there are very few options available to do that at the moment.

One of those options is to the use the OpenSSL ocsp command-line tool, although it's a bit cumbersome to use.

The first step is to retrieve the URL at which the CA provides an OCSP responder. This can be done via the OpenSSL x509 command:

```
openssl x509 -noout -ocsp_uri -in /path/to/the/server/certificate
```

It will output something like "http://ocsp.int-x3.letsencrypt.org".

Now we can use the OCSP tool to request an OCSP response for this certificate from the CA, provided that we have the certificate of the CA at hand, but it's usually needed to get a correct chain of certificates anyway:

```
openssl ocsp -issuer /path/to/the/ca/certificate -cert /path/to/the/server/
↪certificate -text -url url/we/retrieved/earlier -respout /path/to/write/the/OCSP/
↪response
```

If everything goes well, this results in an OCSP response for the server certificate being written to /path/to/write/the/OCSP/response. It seems that earlier versions of OpenSSL did not properly handle the URL, and one needed to split the host and path parts of the OCSP URL, and use the `-header` option of the ocsp command:

```
openssl ocsp -issuer /path/to/the/ca/certificate -cert /path/to/the/server/
↪certificate -text -url <path> -header 'Host' <host> -respout /path/to/write/the/
↪OCSP/response
```

(continues on next page)

(continued from previous page)

We can now use it directly with the *addDOHLocal()* or the *addTLSLocal()* functions:

```
addDOHLocal("127.0.0.1:443", "/path/to/the/server/certificate", "/path/to/the/server/private/key", { "/" }, { ocspResponses={"/path/to/write/the/OCSP/response"}})
addTLSLocal("127.0.0.1:853", "/path/to/the/server/certificate", "/path/to/the/server/private/key", { ocspResponses={"/path/to/write/the/OCSP/response"}})
```

Since this response will be only valid for a while, a script needs to be written to retrieve it regularly via `cron` or any other mechanism. Once the new response has been retrieved, it is possible to tell dnsdist to reload it by connecting to the *console* and calling *reloadAllCertificates()* so that it reloads the certificates, keys and OCSP responses associated to the DNS over TLS and DNS over HTTPS contexts.

18.13.3 Testing

Once a valid OCSP response has retrieved and loaded into dnsdist, it is possible to test that everything is working fine using the OpenSSL s_client command:

```
openssl s_client -connect <IP:port> -status -servername <SNI name to use> | grep -F 'OCSP Response Status'
```

should return something like `OCSP Response Status: successful (0x0)`, indicating that the client received a valid OCSP stapling response from the server.

18.14 TLS Certificates Management

TLS certificates and keys are used in several places of **dnsdist**, dealing with incoming connections over *DNS-over-TLS*, *DNS-over-HTTPS (DoH)*, *DNS-over-HTTP/3 (DoH3)* and *DNS-over-QUIC (DoQ)*.

The related functions (*addTLSLocal()*, *addDOHLocal()*, *addDOH3Local()* and *addDOQLocal()*) accept:

- a path to a X.509 certificate file in `PEM` format, or a list of paths to such files, or a *TLSCertificate* object
- a path to the private key file corresponding to the certificate, or a list of paths to such files whose order should match the certificate files ones. This parameter is ignored if the first one contains *TLSCertificate* objects, as keys are then retrieved from the objects.

For example, to load two certificates, one `RSA` and one `ECDSA` one:

```
addTLSLocal("192.0.2.1:853", { "/path/to/rsa/pem", "/path/to/ecdsa/pem" }, { "/path/to/rsa/key", "/path/to/ecdsa/key" })
```

18.14.1 Password-protected PKCS12 files

Note: `PKCS12` support requires the use of the `openssl` TLS provider.

dnsdist can use password-protected `PKCS12` certificates and keys. The certificate and key are loaded from a password-protected file using *newTLSCertificate()* which returns a *TLSCertificate* object, which can then be passed to *addTLSLocal()*, *addDOHLocal()*, *addDOH3Local()* and *addDOQLocal()*.

```
myCertObject = newTLSCertificate("path/to/domain.p12", {password="passphrase"}) -- use a password protected PKCS12 file
```

18.14.2 Reloading certificates

There are two ways to instruct **dnsdist** to reload the certificate and key files from disk. The easiest one is to use *reloadAllCertificates()* which reload all *DNSCrypt* and TLS certificates, along with their associated keys. The second allows a finer-grained, per-bind, approach:

```
-- reload certificates and keys for DoT binds:
for idx = 0, getTLSFrontendCount() - 1 do
  frontend = getTLSFrontend(idx)
  frontend:reloadCertificates()
end

-- reload certificates and keys for DoH binds:
for idx = 0, getDOHFrontendCount() - 1 do
  frontend = getDOHFrontend(idx)
  frontend:reloadCertificates()
end

-- reload certificates and keys for DoQ binds:
for idx = 0, getDOQFrontendCount() - 1 do
  frontend = getDOQFrontend(idx)
  frontend:reloadCertificates()
end

-- reload certificates and keys for DoH3 binds:
for idx = 0, getDOH3FrontendCount() - 1 do
  frontend = getDOH3Frontend(idx)
  frontend:reloadCertificates()
end
```

18.14.3 TLS sessions

See *TLS Sessions Management*.

18.14.4 OCSP stapling

See *OCSP Stapling*.

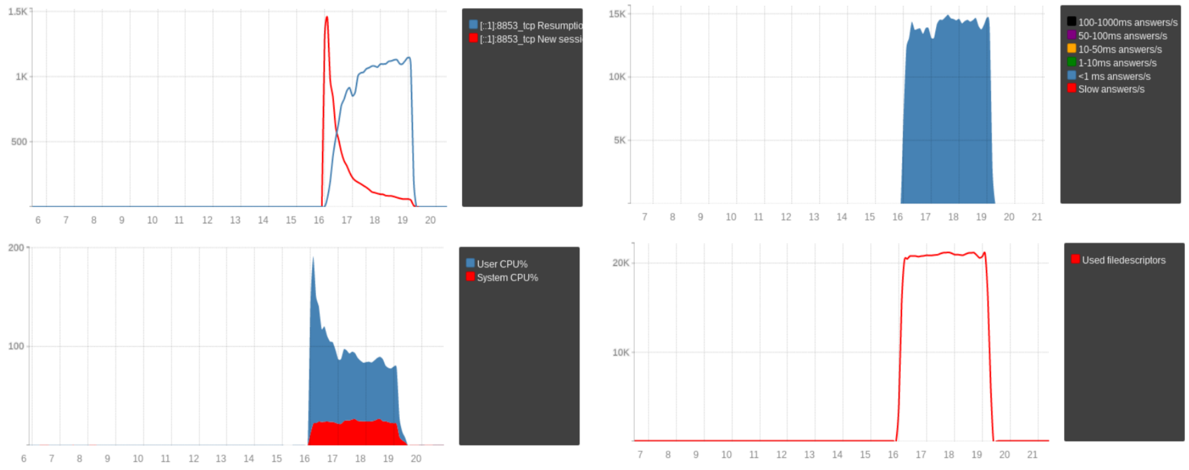
18.15 TLS Sessions Management

18.15.1 TLS sessions

One of the most costly TLS operation is the negotiation of a new session, since both the client and the server need to generate and agree on cryptographic materials. In order to reduce that cost, TLS implements what is called session resumption, where a client opening a new connection to a server can reuse the cryptographic materials negotiated for a previous TLS session.

The following figures show that, with the same number of established incoming connections and queries per second, the ratio of new TLS sessions and resumed sessions has a huge impact on CPU usage:

The necessary information to resume a session can either be kept on the server's side (sessions) or on the client's one (tickets). Initially only the server-side approach existed, with two drawbacks:



- the server needs to keep that information at hand, for a client that might never come back;
- sharing that information between several servers is not easy, especially in setups involving anycast or any kind of cluster without strong session affinity.

Nowadays pretty much all clients support the second option, TLS tickets, where the need information is signed and encrypted by the server before being sent to the client, which is responsible for storing it and sending it back when it wants to establish a new session. That reduces the burden of the server while providing the same benefits.

The server uses Session Ticket Encryption Key (STEK) to sign and encrypt the information sent to the client, making it possible to ensure that it is genuine and has not been tampered when the client provides it later. That STEK can be shared by all dnsdist instances in the same cluster, making it possible for any server to resume a session initially generated by a different server.

Knowing the STEK is all the information needed to be able to decrypt a live TLS session, but also a recorded one, so it is very important to keep that key well-protected. It should never be exchanged in clear-text, and ideally should not be written to persistent storage but be kept in a tmpfs with no swap configured. It should also be regularly rotated to preserve TLS' forward secrecy properties.

18.15.2 Keys management for incoming connections in dnsdist

dnsdist supports both server's side (sessions) and client's side (tickets) resumption for incoming connections (client to dnsdist).

Since server-side sessions cannot be shared between several instances, and pretty much all clients support tickets anyway, we do recommend disabling the sessions by passing `numberOfStoredSessions=0` to the *`addDOHLocal()`* (for DNS over HTTPS) and *`addTLSLocal()`* (for DNS over TLS) functions.

By default, dnsdist will generate a new, random STEK at startup for each *`addTLSLocal()`* and *`addDOHLocal()`* directive, and rotate these STEKs every 12 hours. For each frontend it will keep 5 keys in memory, with only the last one marked as active and used to encrypt new tickets while the remaining ones can still be used to decrypt existing tickets after a rotation. The rotation time and the number of keys to keep in memory can be configured via the `numberOfTicketsKeys` and `ticketsKeysRotationDelay` parameters of the *`addDOHLocal()`* (for DNS over HTTPS) and *`addTLSLocal()`* (for DNS over TLS) functions. When the automatic rotation mechanism kicks in a new, random key will be added to the list of keys. With the OpenSSL provider, the new key becomes active, so new tickets will be encrypted with this key, and the existing keys become passive and only be used to decrypt existing tickets. With the GnuTLS provider only one key is currently supported so the existing keys are immediately discarded. This automatic rotation can be disabled by setting `ticketsKeysRotationDelay` to 0.

It is also possible to manually request a STEK rotation using the *`getDOHFrontend()`* (DoH) and *`getTLSContext()`* (DoT) functions to retrieve the bind object, and calling its `rotateTicketsKey` method (*`DOHFrontend:rotateTicketsKey()`*, *`TLSContext:rotateTicketsKey()`*).

The default settings should be fine for most deployments, but generating a random key for every dnsdist instance will not allow resuming the session from a different instance in a cluster. It is also not very useful to have a different key for every *addTLSLocal()* and *addDOHLocal()* directive if you are using the same certificate and key, and it would be much better to use the same STEK to improve the session resumption ratio.

In that case it is possible to generate the STEK outside of dnsdist, write it to a file, distribute it to all instances using something like rsync over SSH, and load that file from dnsdist. Please remember that the STEK contains very sensitive data, and should be well-protected from access by unauthorized users. It means that special care should be taken to setting the right permissions on that file. Automatic rotation should then be disabled by setting `ticketsKeysRotationDelay` to 0.

For the OpenSSL provider (DoT, DoH), generating a random STEK in a file is a simple as getting 80 cryptographically secure random bytes and writing them to a file:

```
dd if=/dev/urandom of=/secure-tmp-fs/tickets.key bs=80 count=1
```

For the GnuTLS provider (DoT), the operation is the same but requires only 64 cryptographically secure random bytes:

```
dd if=/dev/urandom of=/secure-tmp-fs/tickets.key bs=64 count=1
```

The file can then be loaded at startup by using the `ticketKeyFile` parameter of the *addDOHLocal()* (for DNS over HTTPS) and *addTLSLocal()* (for DNS over TLS) functions.

If the file contains several keys, so for example 240 random bytes, dnsdist will load several STEKs, using the last one for encrypting new tickets and all of them to decrypt existing tickets.

In order to rotate the keys at runtime, it is possible to instruct dnsdist to reload the content of the certificates, keys, and STEKs from the same file used at configuration time, for all DoH and DoH binds, by issuing the *reloadAllCertificates()* command. It can also be done one bind at a time using the *getDOHFrontend()* (DoH) and *getTLSContext()* (DoT) functions to retrieve the bind object, and calling its `loadTicketsKeys` method (*DOHFrontend:loadTicketsKeys()*, *TLSContext:loadTicketsKeys()*).

One possible way of handling manual rotation of the key would be to first:

- generate `N` keys in `N (1..N)` separate files (for example executing `dd if=/dev/urandom of=/secure-tmp-fs/N.key bs=80 count=1` `N` times)
- concatenate the `N` files into a single file (`/secure-tmp-fs/STEKs.key`) that you pass to dnsdist's `ticketKeyFile` parameter

Then, when the STEK should be rotated:

- generate one new key file (`N+1`)
- delete the first key file (`1`)
- concatenate the `2..N+1` files into one (`/secure-tmp-fs/STEKs.key`)
- issue *reloadAllCertificates()* via the dnsdist console, or call `loadTicketsKeys('/secure-tmp-fs/STEKs.key')` for all frontends

This way dnsdist can still decrypt incoming tickets that were encoded via the previous key (the active one is always the one at the end of the file, and we start by removing the one at the beginning of the file).

18.15.3 Content of the STEK file

It does not really matter for most operations, but for later reference the format of the OpenSSL STEK is:

- a 16 bytes binary key identifier
- a 32 bytes AES 256 key
- a 32 bytes HMAC SHA-2 256 key

For GnuTLS:

- a 16 bytes binary key identifier
- a 32 bytes AES 256 key
- a 16 bytes HMAC SHA-1 key

18.15.4 Sessions management for outgoing connections

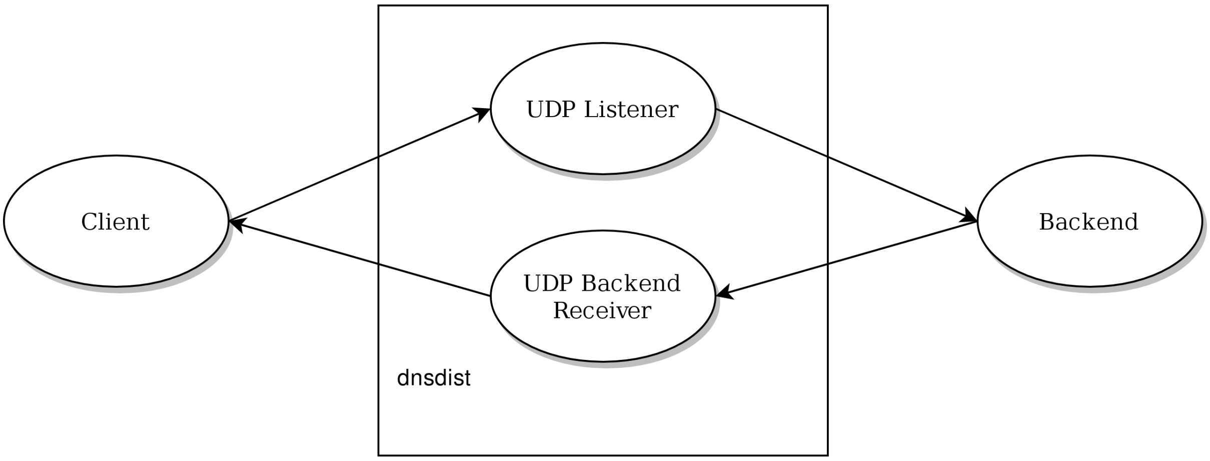
Since 1.7, dnsdist supports securing the connection toward backends using DNS over TLS. For these connections, it keeps a cache of TLS tickets to be able to resume a TLS session quickly. By default that cache contains up to 20 TLS tickets per-backend, is cleaned up every 60s, and TLS tickets expire if they have not been used after 600 seconds. These values can be set at configuration time via:

- `setOutgoingTLSSessionsCacheMaxTicketsPerBackend()`
- `setOutgoingTLSSessionsCacheCleanupDelay()`
- `setOutgoingTLSSessionsCacheMaxTicketValidity()`

18.16 Internal Design

This part of the documentation is intended for developers interested in understanding how the actual code works, and might not be of much interest to regular users.

18.16.1 UDP design

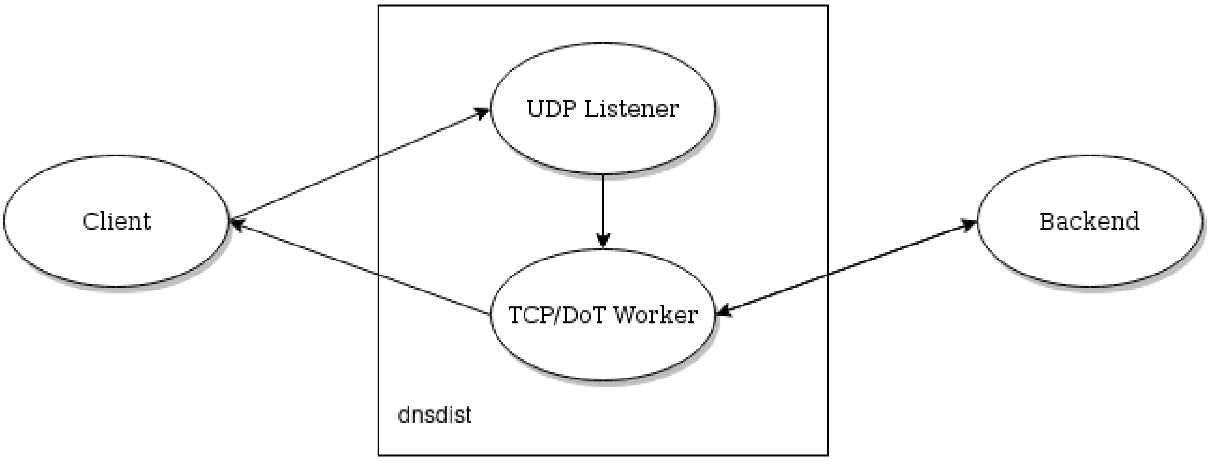


For UDP queries, dnsdist stores the initial ID in a per-backend table called *IDState*. That ID then replaced by one derived from a counter before forwarding the query to the backend, to prevent duplicated IDs sent clients from making it to the backend. When the response is received, dnsdist uses the ID sent by the backend to find the corresponding *IDState* and restores the initial ID, as well as some flags if needed, before sending the response to the client.

That design means that there is a maximum of 65535 in-flight UDP queries per backend. It can actually be even less than that if `setMaxUDPOutstanding()` is set to a lower value, for example to reduce the overall memory usage.

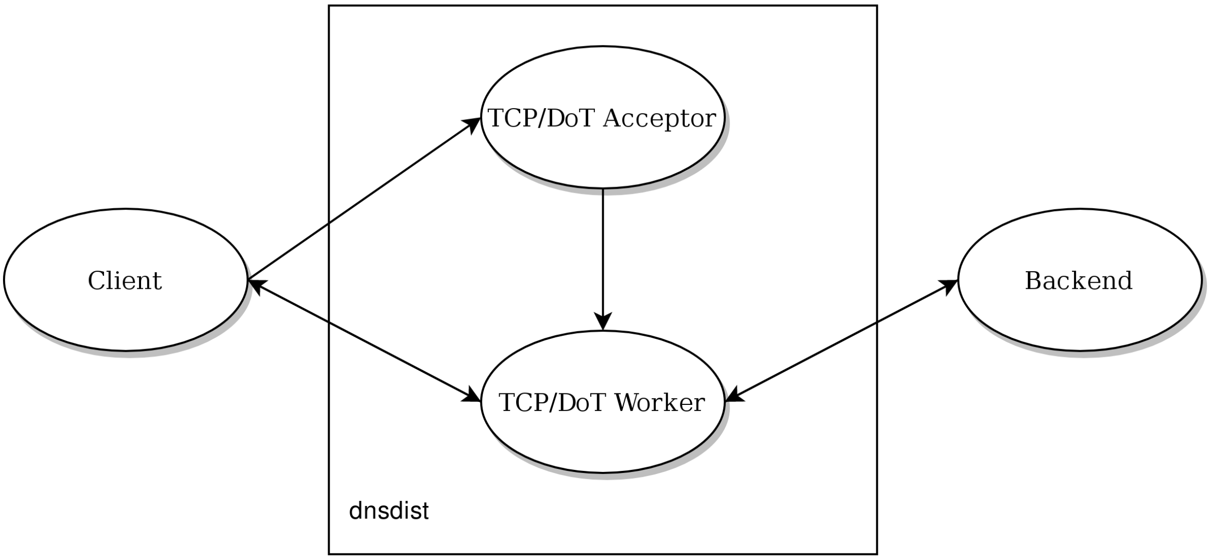
Note that the source address and port used to contact a given backend is set at startup, for performance reasons, and then only changes on reconnect. There might be more than one socket, and thus several ports, if the `sockets` parameter was set to a higher value than 1 on the `newServer()` directive.

Note that, since 1.7, UDP queries can be passed to the backend over TCP if the backend is TCP-only, or configured for DNS over TLS. This is done by passing the incoming query to a TCP worker over a pipe, as was already done for incoming TCP queries.



In that case the response will be sent back, directly by the TCP worker, over UDP, instead of being passed back to the UDP responder thread.

18.16.2 TCP / DoT design

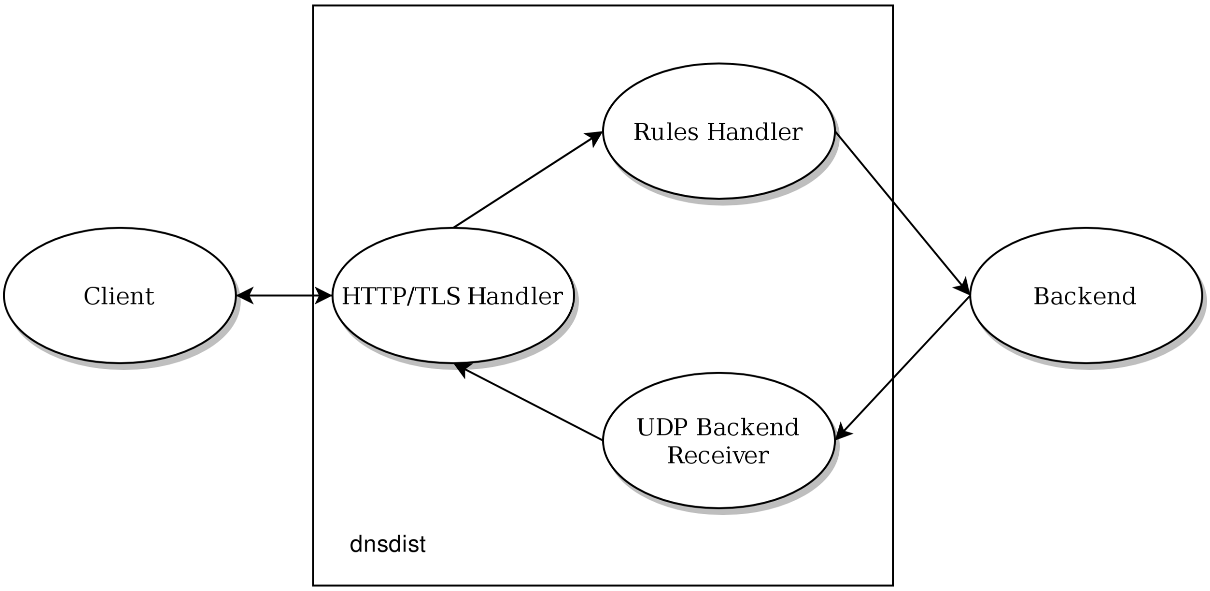


For TCP and DoT, a single thread is created for each `addLocal()` and `addTLSLocal()` directive, listening to the incoming TCP sockets, accepting new connections and distributing them over a pipe to the TCP worker threads. These threads handle both the TCP connection with the client and the one with the backend.

18.16.3 DNS over HTTP/2 design

h2o (up to 1.7)

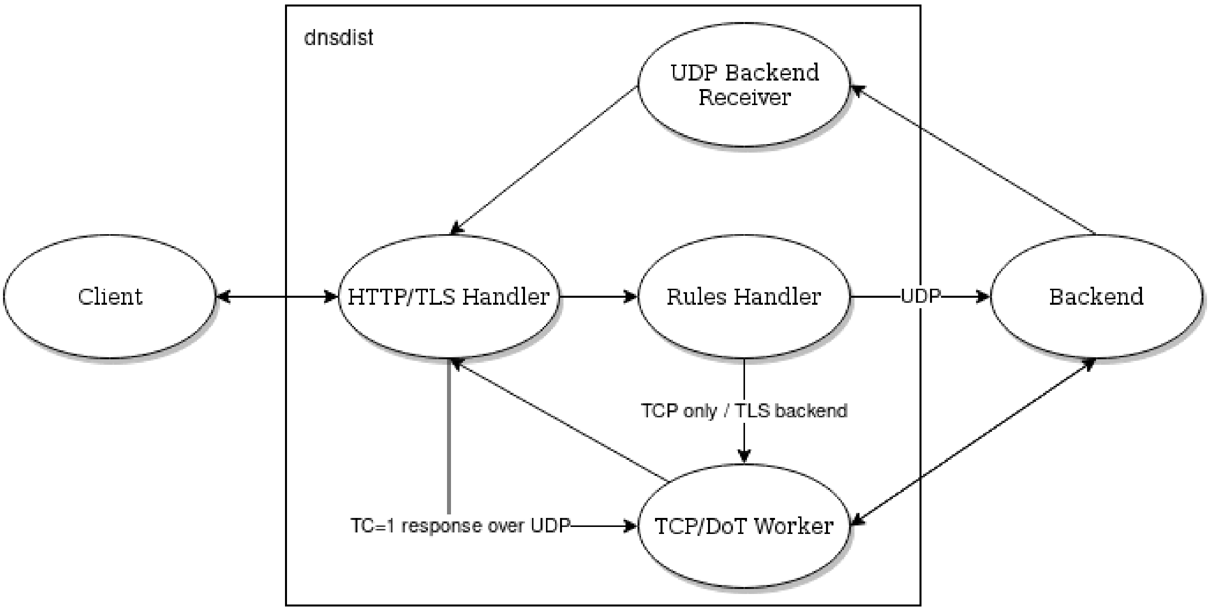
For DNS over HTTP/2, two threads are created for each `addDOHLocal()` directive, one handling the TLS and HTTP layers, then passing the queries to the second one over a pipe. The second thread does DNS processing, applying rules and forwarding the query to the backend if needed, over UDP. Note that even if the query does not need to be passed to a backend (cache-hit, self-generated answer), the response will be passed back to the first thread via a pipe, since only that thread deals with the client. If the response comes from a backend, it will be



picked up by the regular UDP listener for that backend, the corresponding *IDState* object located, and the response sent to the first thread over a pipe.

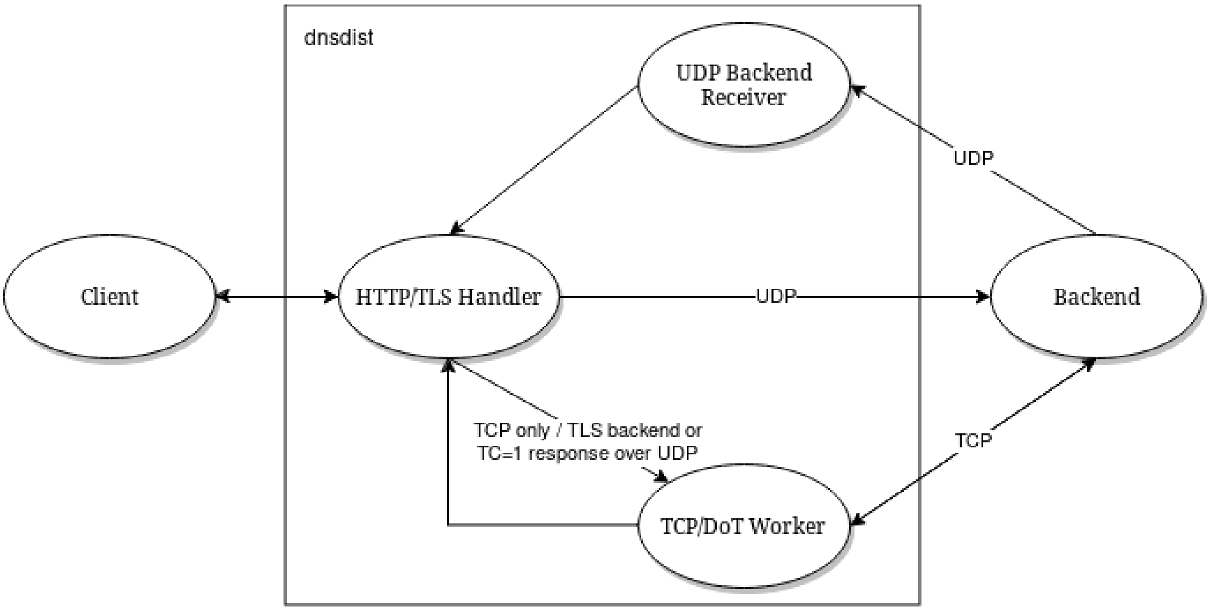
h2o (1.7 - 1.9)

Since 1.7, if the UDP response coming from the backend has been truncated (TC bit is set), dnsdist will retry over TCP by passing the query to a TCP worker over a pipe, as was already done for incoming TCP queries. The response will then be passed back to the DoH worker thread over the same pipe that for UDP queries. That also happens if the backend is marked TCP-only, or configured for DNS over TLS, in which case the query is obviously not sent over UDP first but immediately sent to a TCP worker thread.



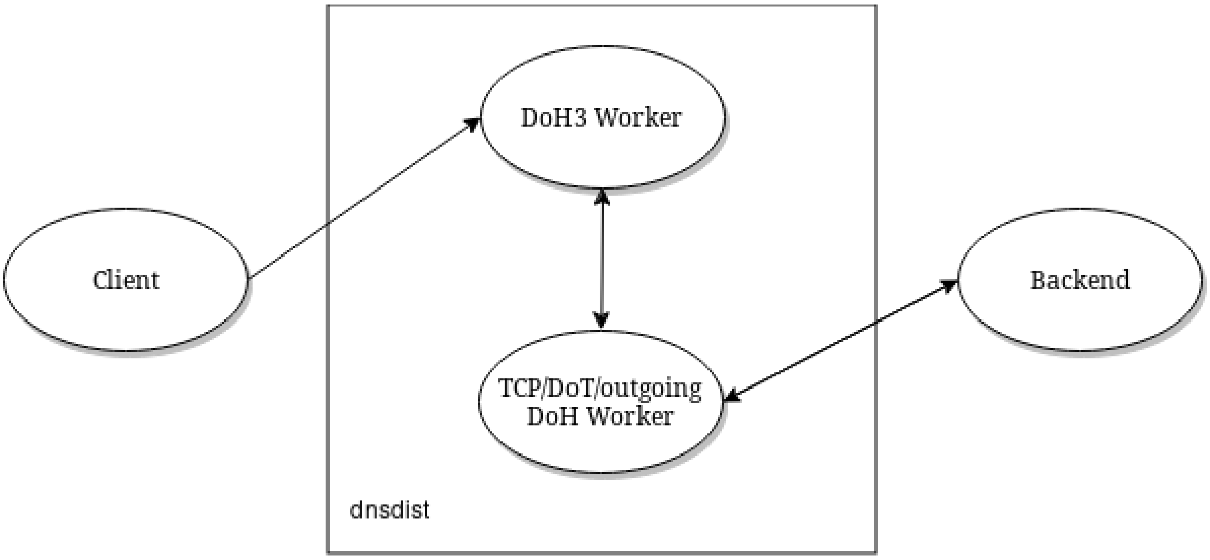
nghttp2 (since 1.9)

Since 1.9 incoming DNS over HTTP/2 is no longer implemented via the `h2o` library but by `nghttp2` instead. The design is roughly the same but has been simplified a bit. As before, if the UDP response coming from the backend has been truncated (TC bit is set), dnsdist will retry over TCP by passing the query to a TCP worker over a pipe, as was already done for incoming TCP queries. The response will then be passed back to the DoH worker thread over the same pipe that for UDP queries. That also happens if the backend is marked TCP-only, or configured for DNS over TLS, in which case the query is obviously not sent over UDP first but immediately sent to a TCP worker thread.



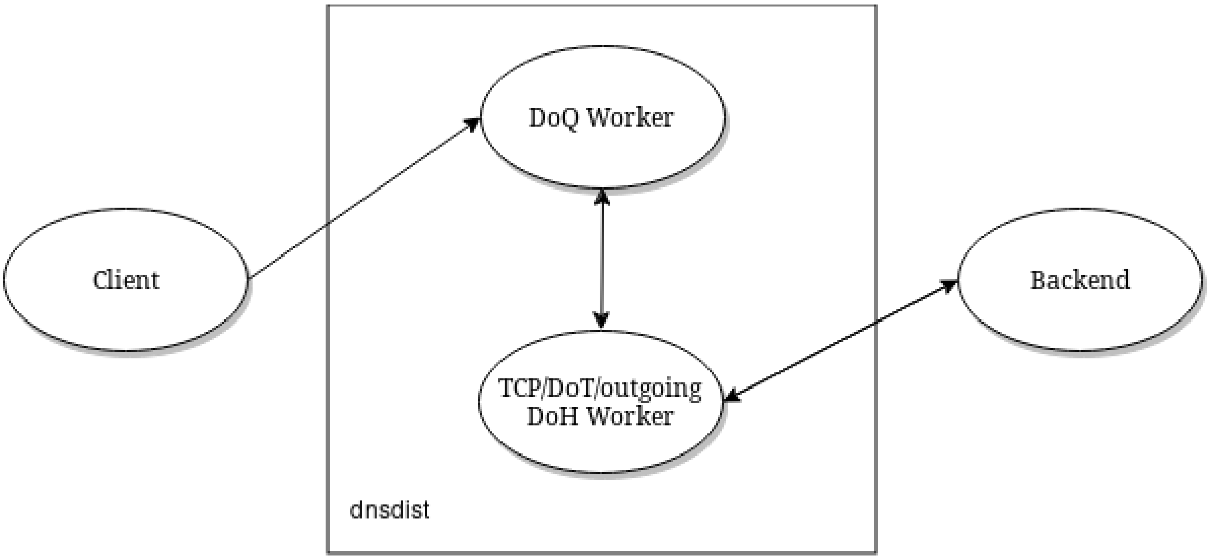
18.16.4 DNS over HTTP/3 design

DNS over HTTP/3 is implemented since 1.9.0 via the `Quiche` library. In 1.9.x, queries received over DNS over HTTP/3 are forwarded to the backend over TCP (Do53 TCP, DoT or DoH2).



18.16.5 DoQ design

DNS over QUIC is implemented since 1.9.0 via the `Quiche` library. In 1.9.x, queries received over DNS over QUIC are forwarded to the backend over TCP (Do53 TCP, DoT or DoH2).

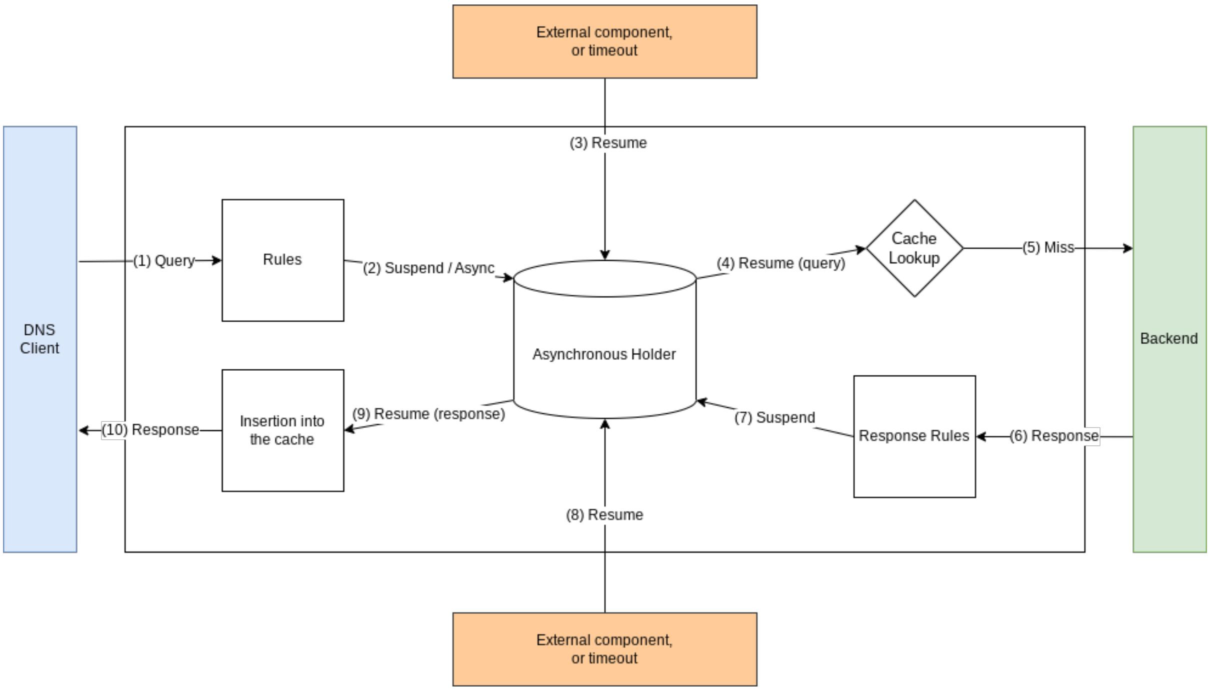


18.17 Asynchronous processing

Since 1.8.0, dnsdist has the ability to process queries and responses in an asynchronous way, suspending them to continue processing other queries and responses, while we are waiting for an external event to occur.

This is done by calling the `DNSQuestion:suspend()` method on a query or a response to pause it, then later the `getAsynchronousObject()` to retrieve it before resuming via `AsynchronousObject:resume()`.

A timeout must be supplied when pausing a query or a response, to prevent paused objects from piling up, consuming memory. When the timeout expires, the suspended object is automatically retrieved and resumes its processing where it was left.



The following code shows a very simple example that forwards queries and responses to an external component over a unix network socket, and resumes them when it gets an answer from the external component.

```
  local asyncID = 0
  local asyncResponderEndpoint = newNetworkEndpoint('/path/to/unix/network/socket/
↪remote/endpoint')
  local listener = newNetworkListener()
  listener:addUnixListeningEndpoint('/path/to/unix/network/socket/local/endpoint',␣
↪0, gotAsyncResponse)
  listener:start()

  function gotAsyncResponse(endpointID, message, from)
    local queryID = tonumber(message)
    local asyncObject = getAsynchronousObject(asyncID, queryID)
    local dq = asyncObject:getDQ()
    dq:setTag(filteringTagName, filteringTagValue)
    asyncObject:resume()
  end

  function passQueryToAsyncFilter(dq)
    local timeout = 500 -- 500 ms
    local buffer = dq:getContent()
    local id = dq.dh:getID()
    dq:suspend(asyncID, id, timeout)
    asyncResponderEndpoint:send(buffer)
    return DNSAction.Allow
  end

function passResponseToAsyncFilter(dr)
    local timeout = 500 -- 500 ms
    local buffer = dr:getContent()
    local id = dr.dh:getID()
    dr:suspend(asyncID, id, timeout)
    asyncResponderEndpoint:send(buffer)
    return DNSResponseAction.Allow
  end

  addAction(AllRule(), LuaAction(passQueryToAsyncFilter))
  addCacheHitResponseAction(AllRule(),␣
↪LuaResponseAction(passResponseToAsyncFilter))
  addResponseAction(AllRule(), LuaResponseAction(passResponseToAsyncFilter))
```

18.18 AF_XDP / XSK

Since 1.9.0, **dnsdist** can use AF_XDP for high performance UDP packet processing recent Linux kernels (4.18+). It requires **dnsdist** to have the `CAP_NET_ADMIN` and `CAP_SYS_ADMIN` capabilities at startup, and to have been compiled with the `--with-xsk` configure option.

Note: To retain the required capabilities it is necessary to call *addCapabilitiesToRetain()* during startup, as **dnsdist** drops capabilities after startup.

Note: `AppArmor` users might need to update their policy to allow **dnsdist** to keep the capabilities. Adding `capability sys_admin,` (for `CAP_SYS_ADMIN`) and `capability net_admin,` (for `CAP_NET_ADMIN`) lines to the policy file is usually enough.

Warning: DNSdist's `AF_XDP` implementation comes with several limitations:

- Asymmetrical network setups where the DNS query and its response do not go through the same network device are not supported
- Ethernet packets larger than 2048 bytes are not supported
- IP and UDP-level checksums are not verified on incoming DNS messages
- IP options in incoming packets are not supported

The way `AF_XDP` works is that **dnsdist** allocates a number of frames in a memory area called a `UMEM`, which is accessible both by the program, in userspace, and by the kernel. Using in-memory ring buffers, the receive (`RX`), transmit (`TX`), completion (`cq`) and fill (`fq`) rings, the kernel can very efficiently pass raw incoming packets to **dnsdist**, which can in return pass raw outgoing packets to the kernel. In addition to these, an `eBPF XDP` program needs to be loaded to decide which packets to distribute via the `AF_XDP` socket (and to which, as there are usually more than one). This program uses a `BPF` map of type `XSKMAP` (located at `/sys/fs/bpf/dnsdist/xskmap` by default) that is populated by **dnsdist** at startup to locate the `AF_XDP` socket to use. **dnsdist** also sets up two additional `BPF` maps (located at `/sys/fs/bpf/dnsdist/xsk-destinations-v4` and `/sys/fs/bpf/dnsdist/xsk-destinations-v6`) to let the `XDP` program know which IP destinations are to be routed to the `AF_XDP` sockets and which are to be passed to the regular network stack (health-checks queries and responses, for example). A ready-to-use XDP program can be found in the `contrib` directory of the PowerDNS Git repository:

```
$ python xdp.py --xsk --interface eth0
```

Then **dnsdist** needs to be configured to use `AF_XDP`, first by creating a *`XskSocket`* object that are tied to a specific queue of a specific network interface:

```
xsk = newXsk({ifName="enp1s0", NIC_queue_id=0, frameNums=65536, xskMapPath="/sys/
↪fs/bpf/dnsdist/xskmap"})
```

This ties the new object to the first receive queue on `enp1s0`, allocating 65536 frames and populating the map located at `/sys/fs/bpf/dnsdist/xskmap`.

Then we can tell **dnsdist** to listen for `AF_XDP` packets to `192.0.2.1:53`, in addition to packets coming via the regular network stack:

```
addLocal("192.0.2.1:53", {xskSocket=xsk})
```

In practice most high-speed (>= 10 Gbps) network interfaces support multiple queues to offer better performance, so we need to allocate one *`XskSocket`* per queue. We can retrieve the number of queues for a given interface via:

```
$ sudo ethtool -l enp1s0
Channel parameters for enp1s0:
Pre-set maximums:
RX:             n/a
TX:             n/a
Other:                  1
Combined:       8
Current hardware settings:
RX:             n/a
TX:             n/a
Other:                  1
Combined:       8
```

The `Combined` lines tell us that the interface supports 8 queues, so we can do something like this:

```
for i=1,8 do
  xsk = newXsk({ifName="enp1s0", NIC_queue_id=i-1, frameNums=65536, xskMapPath="/
↪sys/fs/bpf/dnsdist/xskmap"})
  addLocal("192.0.2.1:53", {xskSocket=xsk, reusePort=true})
```

(continues on next page)

(continued from previous page)

```
end
```

This will start one router thread per `XskSocket` object, plus one worker thread per `addLocal()` using that `XskSocket` object.

We can instructs **dnsdist** to use `AF_XDP` to send and receive UDP packets to a backend in addition to packets from clients:

```
local sockets = {}
for i=1,8 do
  xsk = newXsk({ifName="enp1s0", NIC_queue_id=i-1, frameNums=65536, xskMapPath="/
↪sys/fs/bpf/dnsdist/xskmap"})
  table.insert(sockets, xsk)
  addLocal("192.0.2.1:53", {xskSocket=xsk, reusePort=true})
end

newServer("192.0.2.2:53", {xskSocket=sockets})
```

This will start one router thread per `XskSocket` object, plus one worker thread per `addLocal()`/`newServer()` using that `XskSocket` object.

We are not passing the MAC address of the backend (or the gateway to reach it) directly, so **dnsdist** will try to fetch it from the system MAC address cache. This may not work, in which case we might need to pass explicitly:

```
newServer("192.0.2.2:53", {xskSocket=sockets, MACAddr='00:11:22:33:44:55'})
```

18.18.1 Performance

Using kxdpgun, we can compare the performance of **dnsdist** using the regular network stack and `AF_XDP`.

This test was realized using two Intel E3-1270 with 4 cores (8 threads) running at 3.8 Ghz, using Intel 82599 10 Gbps network cards. On both the injector running `kxdpgun` and the box running **dnsdist** there was no firewall, the governor was set to `performance`, the UDP buffers were raised to `16777216` and the receive queue hash policy set to use the IP addresses and ports (see *Performance Tuning*).

dnsdist was configured to immediately respond to incoming queries with `REFUSED`:

```
addAction(AllRule(), RCodeAction(DNSRCode.REFUSED))
```

On the injector box we executed:

```
$ sudo kxdpgun -Q 2500000 -p 53 -i random_1M 192.0.2.1 -t 60
using interface enp1s0, XDP threads 8, UDP, native mode
[...]
```

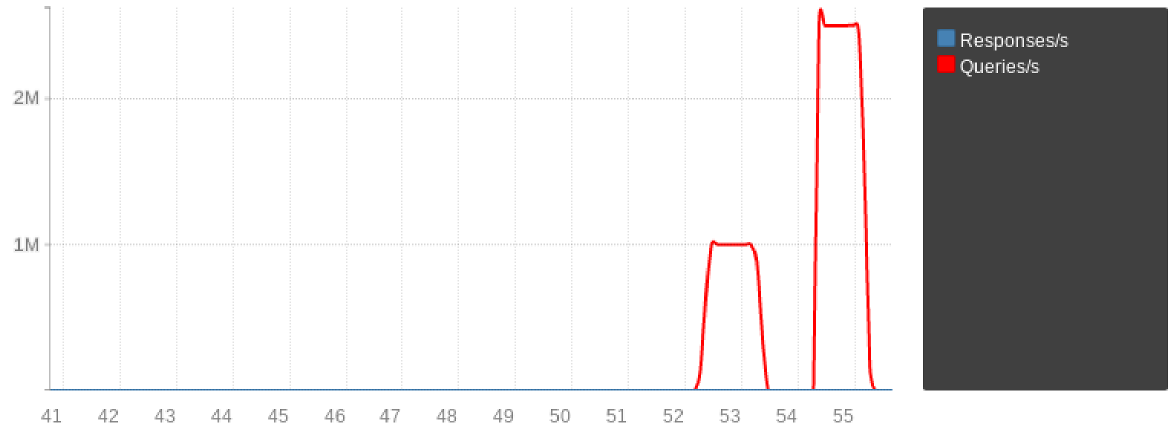
We first ran without `AF_XDP`:

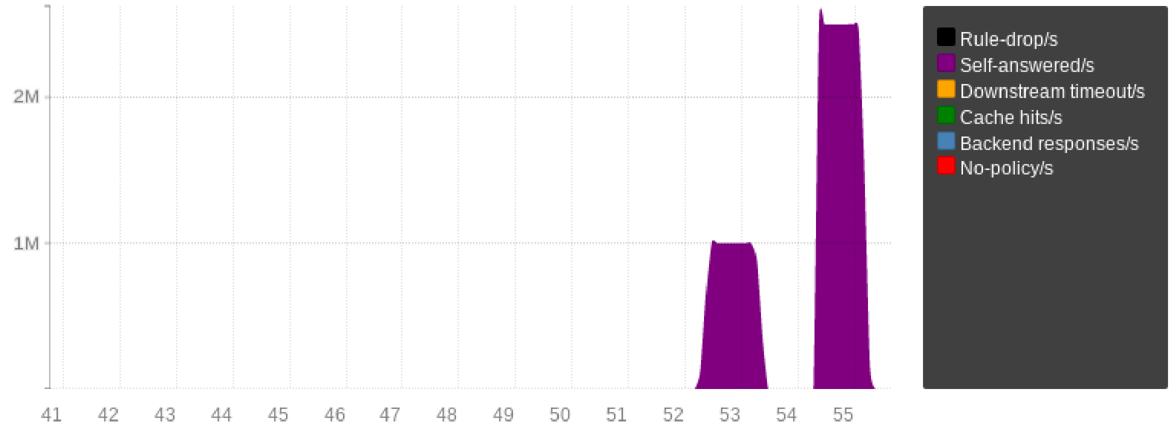
```
for i=1,8 do
  addLocal("192.0.2.1:53", {reusePort=true})
end
```

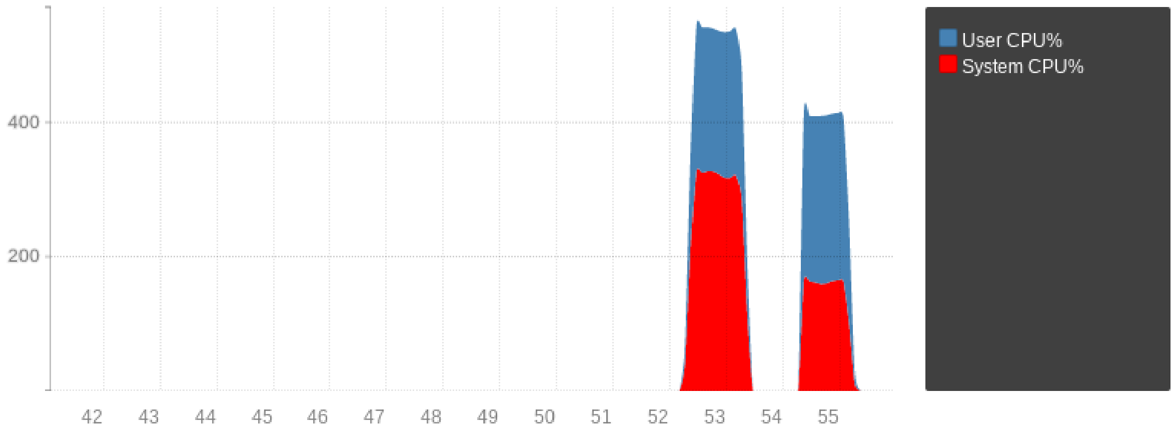
then with:

```
for i=1,8 do
  xsk = newXsk({ifName="enp1s0", NIC_queue_id=i-1, frameNums=65536, xskMapPath="/
↪sys/fs/bpf/dnsdist/xskmap"})
  addLocal("192.0.2.1:53", {xskSocket=xsk, reusePort=true})
end
```

The first run handled roughly 1 million QPS, the second run 2.5 millions, with the CPU usage being much lower in the `AF_XDP` case.


2M
1M
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
Responses/s
Queries/s


2M
1M
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
Rule-drop/s
Self-answered/s
Downstream timeout/s
Cache hits/s
Backend responses/s
No-policy/s


400
200
42
43
44
45
46
47
48
49
50
51
52
53
54
55
User CPU%
System CPU%

CHAPTER

NINETEEN

REFERENCE GUIDES

These chapters contain extensive information on all functions and object available in dnsdist.

19.1 Rule Actions

Rule selectors need to be combined with an action for them to actually do something with the matched packets. Some actions allow further processing of rules, this is noted in their description. Most of these start with 'Set' with a few exceptions, mostly for logging actions. These exceptions are:

- `ClearRecordTypesResponseAction()`
- `KeyValueStoreLookupAction()`
- `DnstapLogAction()`
- `DnstapLogResponseAction()`
- `LimitTTLResponseAction()`
- `LogAction()`
- `LogResponseAction()`
- `NoneAction()`
- `RemoteLogAction()`
- `RemoteLogResponseAction()`
- `SNMPTrapAction()`
- `SNMPTrapResponseAction()`
- `TeeAction()`

The following actions exist.

`AllowAction`()

Let these packets go through.

`AllowResponseAction`()

Let these packets go through.

`ClearRecordTypesResponseAction`(*types*)

New in version 1.8.0.

Removes given type(s) records from the response. Beware you can accidentally turn the answer into a NODATA response without a SOA record in the additional section in which case you may want to use `NegativeAndSOAAction()` to generate an answer, see example below. Subsequent rules are processed after this action.

```
-- removes any HTTPS record in the response
addResponseAction(
        QNameRule('www.example.com.'),
        ClearRecordTypesResponseAction(DNSQType.HTTPS)
)
-- reply directly with NODATA and a SOA record as we know the answer will be empty
addAction(
        AndRule{QNameRule('www.example.com.'), QTypeRule(DNSQType.HTTPS)},
        NegativeAndSOAAction(false, 'example.com.', 3600, 'ns.example.com.', 'postmaster.example.com.', 1, 1800, 900, 604800, 86400)
)
```

Parameters **types** (*int*) – a single type or a list of types to remove

ContinueAction (*action*)

New in version 1.4.0.

Execute the specified action and override its return with None, making it possible to continue the processing. Subsequent rules are processed after this action.

Parameters **action** (*int*) – Any other action

DelayAction (*milliseconds*)

Delay the response by the specified amount of milliseconds (UDP-only). Note that the sending of the query to the backend, if needed, is not delayed. Only the sending of the response to the client will be delayed. Subsequent rules are processed after this action.

Parameters **milliseconds** (*int*) – The amount of milliseconds to delay the response

DelayResponseAction (*milliseconds*)

Delay the response by the specified amount of milliseconds (UDP-only). The only difference between this action and *DelayAction()* is that they can only be applied on, respectively, responses and queries. Subsequent rules are processed after this action.

Parameters **milliseconds** (*int*) – The amount of milliseconds to delay the response

DisableECSAction ()

Deprecated since version 1.6.0.

This function has been deprecated in 1.6.0 and removed in 1.7.0, please use *SetDisableECSAction()* instead.

Disable the sending of ECS to the backend. Subsequent rules are processed after this action.

DisableValidationAction ()

Deprecated since version 1.6.0.

This function has been deprecated in 1.6.0 and removed in 1.7.0, please use *SetDisableValidationAction()* instead.

Set the CD bit in the query and let it go through. Subsequent rules are processed after this action.

DnstapLogAction (*identity*, *logger*[, *alterFunction*])

Send the current query to a remote logger as a *dnstap* message. `alterFunction` is a callback, receiving a *DNSQuestion* and a *DnstapMessage*, that can be used to modify the message. Subsequent rules are processed after this action.

Parameters

- **identity** (*string*) – Server identity to store in the dnstap message
- **logger** – The *FrameStreamLogger* or *RemoteLogger* object to write to
- **alterFunction** – A Lua function to alter the message before sending

DnstapLogResponseAction(*identity*, *logger*[, *alterFunction*])

Send the current response to a remote logger as a *dnstap* message. `alterFunction` is a callback, receiving a `DNSQuestion` and a `DnstapMessage`, that can be used to modify the message. Subsequent rules are processed after this action.

Parameters

- **identity** (*string*) – Server identity to store in the dnstap message
- **logger** – The `FrameStreamLogger` or `RemoteLogger` object to write to
- **alterFunction** – A Lua function to alter the message before sending

DropAction()

Drop the packet.

DropResponseAction()

Drop the packet.

ECSOverrideAction(*override*)

Deprecated since version 1.6.0.

This function has been deprecated in 1.6.0 and removed in 1.7.0, please use `SetECSOverrideAction()` instead.

Whether an existing EDNS Client Subnet value should be overridden (true) or not (false). Subsequent rules are processed after this action.

Parameters **override** (*bool*) – Whether or not to override ECS value

ECSPrefixLengthAction(*v4*, *v6*)

Deprecated since version 1.6.0.

This function has been deprecated in 1.6.0 and removed in 1.7.0, please use `SetECSPrefixLengthAction()` instead.

Set the ECS prefix length. Subsequent rules are processed after this action.

Parameters

- **v4** (*int*) – The IPv4 netmask length
- **v6** (*int*) – The IPv6 netmask length

ERCodeAction(*rcode*[, *options*])

New in version 1.4.0.

Changed in version 1.5.0: Added the optional parameter `options`.

Reply immediately by turning the query into a response with the specified EDNS extended `rcode`. `rcode` can be specified as an integer or as one of the built-in *RCode*.

Parameters

- **rcode** (*int*) – The extended RCODE to respond with.
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `aa`: bool - Set the AA bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ad`: bool - Set the AD bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ra`: bool - Set the RA bit to this value (true means the bit is set, false means it's cleared). Default is to copy the value of the RD bit from the incoming query.

HTTPStatusAction (*status*, *body*, *contentType=""*[, *options*])

New in version 1.4.0.

Changed in version 1.5.0: Added the optional parameter `options`.

Return an HTTP response with a status code of ''status''. For HTTP redirects, ''body'' should be the redirect URL.

Parameters

- **status** (*int*) – The HTTP status code to return.
- **body** (*string*) – The body of the HTTP response, or a URL if the status code is a redirect (3xx).
- **contentType** (*string*) – The HTTP Content-Type header to return for a 200 response, ignored otherwise. Default is ''application/dns-message''.
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `aa`: bool - Set the AA bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ad`: bool - Set the AD bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ra`: bool - Set the RA bit to this value (true means the bit is set, false means it's cleared). Default is to copy the value of the RD bit from the incoming query.

KeyValueStoreLookupAction (*kvs*, *lookupKey*, *destinationTag*)

New in version 1.4.0.

Does a lookup into the key value store referenced by 'kvs' using the key returned by 'lookupKey', and storing the result if any into the tag named 'destinationTag'. The store can be a CDB (*newCDBKVStore()*) or a LMDB database (*newLMDBKVStore()*). The key can be based on the qname (*KeyValueLookupKeyQName()* and *KeyValueLookupKeySuffix()*), source IP (*KeyValueLookupKeySourceIP()*) or the value of an existing tag (*KeyValueLookupKeyTag()*). Subsequent rules are processed after this action. Note that the tag is always created, even if there was no match, but in that case the content is empty.

Parameters

- **kvs** (`KeyValueStore`) – The key value store to query
- **lookupKey** (*KeyValueLookupKey*) – The key to use for the lookup
- **destinationTag** (*string*) – The name of the tag to store the result into

KeyValueStoreRangeLookupAction (*kvs*, *lookupKey*, *destinationTag*)

New in version 1.7.0.

Does a range-based lookup into the key value store referenced by 'kvs' using the key returned by 'lookupKey', and storing the result if any into the tag named 'destinationTag'. This assumes that there is a key in network byte order for the last element of the range (for example 2001:0db8:ffff:ffff:ffff:ffff:ffff:ffff for 2001:db8::/32) which contains the first element of the range (2001:0db8:0000:0000:0000:0000:0000:0000) (optionally followed by any data) as value, also in network byte order, and that there is no overlapping ranges in the database. This requires that the underlying store supports ordered keys, which is true for LMDB but not for CDB.

Subsequent rules are processed after this action.

Parameters

- **kvs** (`KeyValueStore`) – The key value store to query
- **lookupKey** (*KeyValueLookupKey*) – The key to use for the lookup
- **destinationTag** (*string*) – The name of the tag to store the result into

LimitTTLResponseAction (*min*[, *max*[, *types*]])
New in version 1.8.0.

Cap the TTLs of the response to the given boundaries.

Parameters

- **min** (*int*) – The minimum allowed value
- **max** (*int*) – The maximum allowed value
- **of int** (*list*) – The record types to cap the TTL for. Default is empty which means all records will be capped.

LogAction ([*filename*[, *binary*[, *append*[, *buffered*[, *verboseOnly*[, *includeTimestamp*]]]]]])
Changed in version 1.4.0: Added the optional parameters `verboseOnly` and `includeTimestamp`, made `filename` optional.

Changed in version 1.7.0: Added the `reload` method.

Log a line for each query, to the specified `file` if any, to the console (require verbose) if the empty string is given as filename.

If an empty string is supplied in the file name, the logging is done to stdout, and only in verbose mode by default. This can be changed by setting `verboseOnly` to false.

When logging to a file, the `binary` optional parameter specifies whether we log in binary form (default) or in textual form. Before 1.4.0 the binary log format only included the qname and qtype. Since 1.4.0 it includes an optional timestamp, the query ID, qname, qtype, remote address and port.

The `append` optional parameter specifies whether we open the file for appending or truncate each time (default). The `buffered` optional parameter specifies whether writes to the file are buffered (default) or not.

Since 1.7.0 calling the `reload()` method on the object will cause it to close and re-open the log file, for rotation purposes.

Subsequent rules are processed after this action.

Parameters

- **filename** (*string*) – File to log to. Set to an empty string to log to the normal stdout log, this only works when `-v` is set on the command line.
- **binary** (*bool*) – Do binary logging. Default true
- **append** (*bool*) – Append to the log. Default false
- **buffered** (*bool*) – Use buffered I/O. Default true
- **verboseOnly** (*bool*) – Whether to log only in verbose mode when logging to stdout. Default is true
- **includeTimestamp** (*bool*) – Whether to include a timestamp for every entry. Default is false

LogResponseAction (*[filename[, append[, buffered[, verboseOnly[, includeTimestamp]]]]]*)
New in version 1.5.0.

Changed in version 1.7.0: Added the `reload` method.

Log a line for each response, to the specified `file` if any, to the console (require verbose) if the empty string is given as filename.

If an empty string is supplied in the file name, the logging is done to stdout, and only in verbose mode by default. This can be changed by setting `verboseOnly` to false.

The `append` optional parameter specifies whether we open the file for appending or truncate each time (default). The `buffered` optional parameter specifies whether writes to the file are buffered (default) or not.

Since 1.7.0 calling the `reload()` method on the object will cause it to close and re-open the log file, for rotation purposes.

Subsequent rules are processed after this action.

Parameters

- **`filename`** (*`string`*) – File to log to. Set to an empty string to log to the normal stdout log, this only works when `-v` is set on the command line.
- **`append`** (*`bool`*) – Append to the log. Default false
- **`buffered`** (*`bool`*) – Use buffered I/O. Default true
- **`verboseOnly`** (*`bool`*) – Whether to log only in verbose mode when logging to stdout. Default is true
- **`includeTimestamp`** (*`bool`*) – Whether to include a timestamp for every entry. Default is false

`LuaAction` (*function*)

Invoke a Lua function that accepts a *`DNSQuestion`*.

The `function` should return a *DNSAction*. If the Lua code fails, ServFail is returned.

Parameters **`function`** (*`string`*) – the name of a Lua function

`LuaFFIAction` (*function*)

New in version 1.5.0.

Invoke a Lua FFI function that accepts a pointer to a `dnsdist_ffi_dnsquestion_t` object, whose bindings are defined in `dnsdist-lua-ffi.hh`.

The `function` should return a *DNSAction*. If the Lua code fails, ServFail is returned.

Parameters **`function`** (*`string`*) – the name of a Lua function

`LuaFFIPerThreadAction` (*function*)

New in version 1.7.0.

Invoke a Lua FFI function that accepts a pointer to a `dnsdist_ffi_dnsquestion_t` object, whose bindings are defined in `dnsdist-lua-ffi.hh`.

The `function` should return a *DNSAction*. If the Lua code fails, ServFail is returned.

The function will be invoked in a per-thread Lua state, without access to the global Lua state. All constants (*DNSQType*, *RCode*, …) are available in that per-thread context, as well as all FFI functions. Objects and their bindings that are not usable in a FFI context (*`DNSQuestion`*, *`DNSDistProtoBufMessage`*, *`PacketCache`*, …) are not available.

Parameters **`function`** (*`string`*) – a Lua string returning a Lua function

`LuaFFIPerThreadResponseAction` (*function*)

New in version 1.7.0.

Invoke a Lua FFI function that accepts a pointer to a `dnsdist_ffi_dnsquestion_t` object, whose bindings are defined in `dnsdist-lua-ffi.hh`.

The `function` should return a *DNSResponseAction*. If the Lua code fails, ServFail is returned.

The function will be invoked in a per-thread Lua state, without access to the global Lua state. All constants (*DNSQType*, *RCode*, …) are available in that per-thread context, as well as all FFI functions. Objects and their bindings that are not usable in a FFI context (*`DNSQuestion`*, *`DNSDistProtoBufMessage`*, *`PacketCache`*, …) are not available.

Parameters **`function`** (*`string`*) – a Lua string returning a Lua function

`LuaFFIResponseAction` (*function*)

New in version 1.5.0.

Invoke a Lua FFI function that accepts a pointer to a `dnsdist_ffi_dnsquestion_t` object, whose bindings are defined in `dnsdist-lua-ffi.hh`.

The `function` should return a *DNSResponseAction*. If the Lua code fails, ServFail is returned.

Parameters **`function`** (*`string`*) – the name of a Lua function

`LuaResponseAction` (*function*)

Invoke a Lua function that accepts a *`DNSResponse`*.

The `function` should return a *DNSResponseAction*. If the Lua code fails, ServFail is returned.

Parameters **`function`** (*`string`*) – the name of a Lua function

`MacAddrAction` (*option*)

Deprecated since version 1.6.0.

This function has been deprecated in 1.6.0 and removed in 1.7.0, please use *`SetMacAddrAction()`* instead.

Add the source MAC address to the query as EDNS0 option `option`. This action is currently only supported on Linux. Subsequent rules are processed after this action.

Parameters **`option`** (*`int`*) – The EDNS0 option number

`NegativeAndSOAAction` (*nxd*, *zone*, *ttl*, *mname*, *rname*, *serial*, *refresh*, *retry*, *expire*, *minimum*[, *options*])

New in version 1.6.0.

Changed in version 1.8.0: Added the `soaInAuthoritySection` option.

Turn a question into a response, either a NXDOMAIN or a NODATA one based on ''nxd”, setting the QR bit to 1 and adding a SOA record in the additional section. Note that this function was called *`SetNegativeAndSOAAction()`* before 1.6.0.

Parameters

- **`nxd`** (*`bool`*) – Whether the answer is a NXDOMAIN (true) or a NODATA (false)
- **`zone`** (*`string`*) – The owner name for the SOA record
- **`ttl`** (*`int`*) – The TTL of the SOA record
- **`mname`** (*`string`*) – The mname of the SOA record
- **`rname`** (*`string`*) – The rname of the SOA record
- **`serial`** (*`int`*) – The value of the serial field in the SOA record
- **`refresh`** (*`int`*) – The value of the refresh field in the SOA record
- **`retry`** (*`int`*) – The value of the retry field in the SOA record
- **`expire`** (*`int`*) – The value of the expire field in the SOA record
- **`minimum`** (*`int`*) – The value of the minimum field in the SOA record
- **`options`** (*`table`*) – A table with key: value pairs with options

Options:

- `aa`: bool - Set the AA bit to this value (true means the bit is set, false means it’s cleared). Default is to clear it.
- `ad`: bool - Set the AD bit to this value (true means the bit is set, false means it’s cleared). Default is to clear it.
- `ra`: bool - Set the RA bit to this value (true means the bit is set, false means it’s cleared). Default is to copy the value of the RD bit from the incoming query.

- `soaInAuthoritySection`: bool - Place the SOA record in the authority section for a complete NXDOMAIN/NODATA response that works as a cacheable negative response, rather than the RPZ-style response with a purely informational SOA in the additional section. Default is false (SOA in additional section).

NoneAction()

Does nothing. Subsequent rules are processed after this action.

NoRecurseAction()

Deprecated since version 1.6.0.

This function has been deprecated in 1.6.0 and removed in 1.7.0, please use *SetNoRecurseAction()* instead.

Strip RD bit from the question, let it go through. Subsequent rules are processed after this action.

PoolAction(*poolname*[, *stop*])

Changed in version 1.8.0: Added the `stop` optional parameter.

Send the packet into the specified pool. If `stop` is set to false, subsequent rules will be processed after this action.

Parameters

- **poolname** (*string*) – The name of the pool
- **stop** (*bool*) – Whether to stop processing rules after this action. Default is true, meaning the remaining rules will not be processed.

QPSAction(*maxqps*)

Drop a packet if it does exceed the `maxqps` queries per second limits. Letting the subsequent rules apply otherwise.

Parameters **maxqps** (*int*) – The QPS limit

QPSPoolAction(*maxqps*, *poolname*[, *stop*])

Changed in version 1.8.0: Added the `stop` optional parameter.

Send the packet into the specified pool only if it does not exceed the `maxqps` queries per second limits. If `stop` is set to false, subsequent rules will be processed after this action. Letting the subsequent rules apply otherwise.

Parameters

- **maxqps** (*int*) – The QPS limit for that pool
- **poolname** (*string*) – The name of the pool
- **stop** (*bool*) – Whether to stop processing rules after this action. Default is true, meaning the remaining rules will not be processed.

RCodeAction(*rcode*[, *options*])

Changed in version 1.5.0: Added the optional parameter `options`.

Reply immediately by turning the query into a response with the specified `rcode`. `rcode` can be specified as an integer or as one of the built-in *RCode*.

Parameters

- **rcode** (*int*) – The RCODE to respond with.
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `aa`: bool - Set the AA bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ad`: bool - Set the AD bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.

- `ra`: bool - Set the RA bit to this value (true means the bit is set, false means it's cleared). Default is to copy the value of the RD bit from the incoming query.

RemoteLogAction(*remoteLogger*[, *alterFunction*[, *options*[, *metas*]]])

Changed in version 1.4.0: `ipEncryptKey` optional key added to the options table.

Changed in version 1.8.0: `metas` optional parameter added. `exportTags` optional key added to the options table.

Send the content of this query to a remote logger via Protocol Buffer. `alterFunction` is a callback, receiving a *DNSQuestion* and a *DNSDistProtoBufMessage*, that can be used to modify the Protocol Buffer content, for example for anonymization purposes. Since 1.8.0 it is possible to add configurable meta-data fields to the Protocol Buffer message via the `metas` parameter, which takes a list of `name``=``key` pairs. For each entry in the list, a new value named `name` will be added to the message with the value corresponding to the `key`. Available keys are:

- `doh-header:<HEADER>`: the content of the corresponding `<HEADER>` HTTP header for DoH queries, empty otherwise
- `doh-host`: the `Host` header for DoH queries, empty otherwise
- `doh-path`: the HTTP path for DoH queries, empty otherwise
- `doh-query-string`: the HTTP query string for DoH queries, empty otherwise
- `doh-scheme`: the HTTP scheme for DoH queries, empty otherwise
- `pool`: the currently selected pool of servers
- `proxy-protocol-value:<TYPE>`: the content of the proxy protocol value of type `<TYPE>`, if any
- `proxy-protocol-values`: the content of all proxy protocol values as a "<type1>:<value1>", …, "<typeN>:<valueN>" strings
- `b64-content`: the base64-encoded DNS payload of the current query
- `sni`: the Server Name Indication value for queries received over DoT or DoH. Empty otherwise.
- `tag:<TAG>`: the content of the corresponding `<TAG>` if any
- `tags`: the list of all tags, and their values, as a "<key1>:<value1>", …, "<keyN>:<valueN>" strings. Note that a tag with an empty value will be exported as "<key>", not "<key>:".

Subsequent rules are processed after this action.

Parameters

- **remoteLogger** (*string*) – The *remoteLogger* object to write to
- **alterFunction** (*string*) – Name of a function to modify the contents of the logs before sending
- **options** (*table*) – A table with key: value pairs.
- **metas** (*table*) – A list of `name``=``key` pairs, for meta-data to be added to Protocol Buffer message.

Options:

- `serverID=""`: str - Set the Server Identity field.
- `ipEncryptKey=""`: str - A key, that can be generated via the *makeIPCipherKey()* function, to encrypt the IP address of the requestor for anonymization purposes. The encryption is done using ipcrypt for IPv4 and a 128-bit AES ECB operation for IPv6.
- `exportTags=""`: str - The comma-separated list of keys of internal tags to export into the `tags` Protocol Buffer field, as "key:value" strings. Note that a tag with an empty value will be exported as "<key>", not "<key>:". An empty string means that no internal tag will be exported. The special value `*` means that all tags will be exported.

RemoteLogResponseAction(*remoteLogger*[, *alterFunction*[, *includeCNAME*[, *options*[, *metas*]]]])

Changed in version 1.4.0: `ipEncryptKey` optional key added to the options table.

Changed in version 1.8.0: `metas` optional parameter added. `exportTags` optional key added to the options table.

Changed in version 1.9.0: `exportExtendedErrorsToMeta` optional key added to the options table.

Send the content of this response to a remote logger via Protocol Buffer. `alterFunction` is the same callback that receiving a *DNSQuestion* and a *DNSDistProtoBufMessage*, that can be used to modify the Protocol Buffer content, for example for anonymization purposes. `includeCNAME` indicates whether CNAME records inside the response should be parsed and exported. The default is to only exports A and AAAA records. Since 1.8.0 it is possible to add configurable meta-data fields to the Protocol Buffer message via the `metas` parameter, which takes a list of `name``=``key` pairs. See *RemoteLogAction()* for the list of available keys. Subsequent rules are processed after this action.

Parameters

- **remoteLogger** (*string*) – The *remoteLogger* object to write to
- **alterFunction** (*string*) – Name of a function to modify the contents of the logs before sending
- **includeCNAME** (*bool*) – Whether or not to parse and export CNAMEs. Default false
- **options** (*table*) – A table with key: value pairs.
- **metas** (*table*) – A list of `name``=``key` pairs, for meta-data to be added to Protocol Buffer message.

Options:

- `serverID=""`: str - Set the Server Identity field.
- `ipEncryptKey=""`: str - A key, that can be generated via the *makeIPCipherKey()* function, to encrypt the IP address of the requestor for anonymization purposes. The encryption is done using ipcrypt for IPv4 and a 128-bit AES ECB operation for IPv6.
- `exportTags=""`: str - The comma-separated list of keys of internal tags to export into the `tags` Protocol Buffer field, as “key:value” strings. Note that a tag with an empty value will be exported as “<key>”, not “<key>:”. An empty string means that no internal tag will be exported. The special value `*` means that all tags will be exported.
- `exportExtendedErrorsToMeta=""`: str - Export Extended DNS Errors present in the DNS response, if any, into the `meta` Protocol Buffer field using the specified `key`. The EDE info code will be exported as an integer value, and the EDE extra text, if present, as a string value.

SetAdditionalProxyProtocolValueAction(*type*, *value*)

New in version 1.6.0.

Add a Proxy-Protocol Type-Length value to be sent to the server along with this query. It does not replace any existing value with the same type but adds a new value. Be careful that Proxy Protocol values are sent once at the beginning of the TCP connection for TCP and DoT queries. That means that values received on an incoming TCP connection will be inherited by subsequent queries received over the same incoming TCP connection, if any, but values set to a query will not be inherited by subsequent queries. Subsequent rules are processed after this action.

Parameters

- **type** (*int*) – The type of the value to send, ranging from 0 to 255 (both included)
- **value** (*str*) – The binary-safe value

SetDisableECSAction()

New in version 1.6.0.

Disable the sending of ECS to the backend. Subsequent rules are processed after this action. Note that this function was called *DisableECSAction()* before 1.6.0.

SetDisableValidationAction()

New in version 1.6.0.

Set the CD bit in the query and let it go through. Subsequent rules are processed after this action. Note that this function was called `DisableValidationAction()` before 1.6.0.

SetECSAction(*v4*[, *v6*])

Set the ECS prefix and prefix length sent to backends to an arbitrary value. If both IPv4 and IPv6 masks are supplied the IPv4 one will be used for IPv4 clients and the IPv6 one for IPv6 clients. Otherwise the first mask is used for both, and can actually be an IPv6 mask. Subsequent rules are processed after this action.

Parameters

- **v4** (*string*) – The IPv4 netmask, for example "192.0.2.1/32"
- **v6** (*string*) – The IPv6 netmask, if any

SetECSOverrideAction(*override*)

New in version 1.6.0.

Whether an existing EDNS Client Subnet value should be overridden (true) or not (false). Subsequent rules are processed after this action. Note that this function was called `ECSOverrideAction()` before 1.6.0.

Parameters **override** (*bool*) – Whether or not to override ECS value

SetECSPrefixLengthAction(*v4*, *v6*)

New in version 1.6.0.

Set the ECS prefix length. Subsequent rules are processed after this action. Note that this function was called `ECSPrefixLengthAction()` before 1.6.0.

Parameters

- **v4** (*int*) – The IPv4 netmask length
- **v6** (*int*) – The IPv6 netmask length

SetEDNSOptionAction(*option*)

New in version 1.7.0.

Add arbitrary EDNS option and data to the query. Any existing EDNS content with the same option code will be overwritten. Subsequent rules are processed after this action.

Parameters

- **option** (*int*) – The EDNS option number
- **data** (*string*) – The EDNS0 option raw content

SetExtendedDNSErrorAction(*infoCode*[, *extraText*])

New in version 1.9.0.

Set an Extended DNS Error status that will be added to the response corresponding to the current query. Subsequent rules are processed after this action.

Parameters

- **infoCode** (*int*) – The EDNS Extended DNS Error code
- **extraText** (*string*) – The optional EDNS Extended DNS Error extra text

SetExtendedDNSErrorResponseAction(*infoCode*[, *extraText*])

New in version 1.9.0.

Set an Extended DNS Error status that will be added to this response. Subsequent rules are processed after this action.

Parameters

- **infoCode** (*int*) – The EDNS Extended DNS Error code
- **extraText** (*string*) – The optional EDNS Extended DNS Error extra text

SetMacAddrAction(*option*)

New in version 1.6.0.

Add the source MAC address to the query as EDNS0 option `option`. This action is currently only supported on Linux. Subsequent rules are processed after this action. Note that this function was called `MacAddrAction()` before 1.6.0.

Parameters **option** (*int*) – The EDNS0 option number

SetMaxReturnedTTLAction(*max*)

New in version 1.8.0.

Cap the TTLs of the response to the given maximum, but only after inserting the response into the packet cache with the initial TTL values.

Parameters **max** (*int*) – The maximum allowed value

SetMaxReturnedTTLResponseAction(*max*)

New in version 1.8.0.

Cap the TTLs of the response to the given maximum, but only after inserting the response into the packet cache with the initial TTL values.

Parameters **max** (*int*) – The maximum allowed value

SetMaxTTLResponseAction(*max*)

New in version 1.8.0.

Cap the TTLs of the response to the given maximum.

Parameters **max** (*int*) – The maximum allowed value

SetMinTTLResponseAction(*min*)

New in version 1.8.0.

Cap the TTLs of the response to the given minimum.

Parameters **min** (*int*) – The minimum allowed value

SetNoRecurseAction()

New in version 1.6.0.

Strip RD bit from the question, let it go through. Subsequent rules are processed after this action. Note that this function was called `NoRecurseAction()` before 1.6.0.

SetNegativeAndSOAAction(*nxd*, *zone*, *ttl*, *mname*, *rname*, *serial*, *refresh*, *retry*, *expire*, *minimum*[, *options*])

New in version 1.5.0.

Deprecated since version 1.6.0.

This function has been deprecated in 1.6.0 and removed in 1.7.0, please use `NegativeAndSOAAction()` instead.

Turn a question into a response, either a NXDOMAIN or a NODATA one based on ‘’nxd”, setting the QR bit to 1 and adding a SOA record in the additional section.

Parameters

- **nxd** (*bool*) – Whether the answer is a NXDOMAIN (true) or a NODATA (false)
- **zone** (*string*) – The owner name for the SOA record
- **ttl** (*int*) – The TTL of the SOA record
- **mname** (*string*) – The mname of the SOA record
- **rname** (*string*) – The rname of the SOA record
- **serial** (*int*) – The value of the serial field in the SOA record
- **refresh** (*int*) – The value of the refresh field in the SOA record

- **retry** (*int*) – The value of the retry field in the SOA record
- **expire** (*int*) – The value of the expire field in the SOA record
- **minimum** (*int*) – The value of the minimum field in the SOA record
- **options** (*table*) – A table with key: value pairs with options

Options:

- `aa`: bool - Set the AA bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ad`: bool - Set the AD bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ra`: bool - Set the RA bit to this value (true means the bit is set, false means it's cleared). Default is to copy the value of the RD bit from the incoming query.

SetProxyProtocolValuesAction(*values*)

New in version 1.5.0.

Set the Proxy-Protocol Type-Length values to be sent to the server along with this query to `values`. Subsequent rules are processed after this action.

Parameters **values** (*table*) – A table of types and values to send, for example: `{ [0] = foo", [42] = "bar" }`

SetReducedTTLResponseAction(*percentage*)

New in version 1.8.0.

Reduce the TTL of records in a response to a percentage of the original TTL. For example, passing 50 means that the original TTL will be cut in half. Subsequent rules are processed after this action.

Parameters **percentage** (*int*) – The percentage to use

SetSkipCacheAction()

New in version 1.6.0.

Don't lookup the cache for this query, don't store the answer. Subsequent rules are processed after this action. Note that this function was called `SkipCacheAction()` before 1.6.0.

SetSkipCacheResponseAction()

New in version 1.6.0.

Don't store this answer into the cache. Subsequent rules are processed after this action.

SetTagAction(*name*, *value*)

New in version 1.6.0.

Changed in version 1.7.0: Prior to 1.7.0 `SetTagAction()` would not overwrite an existing tag value if already set.

Associate a tag named `name` with a value of `value` to this query, that will be passed on to the response. This function will overwrite any existing tag value. Subsequent rules are processed after this action. Note that this function was called `TagAction()` before 1.6.0.

Parameters

- **name** (*string*) – The name of the tag to set
- **value** (*string*) – The value of the tag

SetTagResponseAction(*name*, *value*)

New in version 1.6.0.

Changed in version 1.7.0: Prior to 1.7.0 `SetTagResponseAction()` would not overwrite an existing tag value if already set.

Associate a tag named `name` with a value of `value` to this response. This function will overwrite any existing tag value. Subsequent rules are processed after this action. Note that this function was called *`TagResponseAction()`* before 1.6.0.

Parameters

- **`name`** (*`string`*) – The name of the tag to set
- **`value`** (*`string`*) – The value of the tag

`SetTempFailureCacheTTLAction`(*ttl*)

New in version 1.6.0.

Set the cache TTL to use for ServFail and Refused replies. TTL is not applied for successful replies. Subsequent rules are processed after this action. Note that this function was called *`TempFailureCacheTTLAction()`* before 1.6.0.

Parameters **`ttl`** (*`int`*) – Cache TTL for temporary failure replies

`SkipCacheAction`()

Deprecated since version 1.6.0.

This function has been deprecated in 1.6.0 and removed in 1.7.0, please use `SetSkipAction()` instead.

Don't lookup the cache for this query, don't store the answer. Subsequent rules are processed after this action.

`SNMPTrapAction`([*message*])

Send an SNMP trap, adding the optional `message` string as the query description. Subsequent rules are processed after this action.

Parameters **`message`** (*`string`*) – The message to include

`SNMPTrapResponseAction`([*message*])

Send an SNMP trap, adding the optional `message` string as the query description. Subsequent rules are processed after this action.

Parameters **`message`** (*`string`*) – The message to include

`SpoofAction`(*ip*[, *options*])
`SpoofAction`(*ips*[, *options*])

Changed in version 1.5.0: Added the optional parameter `options`.

Changed in version 1.6.0: Up to 1.6.0, the syntax for this function was `SpoofAction(ips[, ip[, options]])`.

Forge a response with the specified IPv4 (for an A query) or IPv6 (for an AAAA) addresses. If you specify multiple addresses, all that match the query type (A, AAAA or ANY) will get spoofed in.

Parameters

- **`ip`** (*`string`*) – An IPv4 and/or IPv6 address to spoof
- **`ips`** (*`{string}`*) – A table of IPv4 and/or IPv6 addresses to spoof
- **`options`** (*`table`*) – A table with key: value pairs with options.

Options:

- `aa`: bool - Set the AA bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ad`: bool - Set the AD bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ra`: bool - Set the RA bit to this value (true means the bit is set, false means it's cleared). Default is to copy the value of the RD bit from the incoming query.
- `ttl`: int - The TTL of the record.

SpoofCNAMEAction(*cname*[, *options*])

Changed in version 1.5.0: Added the optional parameter `options`.

Forge a response with the specified CNAME value.

Parameters

- **cname** (*string*) – The name to respond with
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `aa`: bool - Set the AA bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ad`: bool - Set the AD bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ra`: bool - Set the RA bit to this value (true means the bit is set, false means it's cleared). Default is to copy the value of the RD bit from the incoming query.
- `ttl`: int - The TTL of the record.

SpoofRawAction(*rawAnswer*[, *options*])
SpoofRawAction(*rawAnswers*[, *options*])

New in version 1.5.0.

Changed in version 1.6.0: Up to 1.6.0, it was only possible to spoof one answer.

Changed in version 1.9.0: Added the optional parameter `typeForAny`.

Forge a response with the specified raw bytes as record data.

```
-- select queries for the 'raw.powerdns.com.' name and TXT type, and answer with both a "aaa" "bbbb" and "ccc" TXT record:
addAction(AndRule({QNameRule('raw.powerdns.com.'), QTypeRule(DNSQType.TXT)}), SpoofRawAction({"\003aaa\004bbbb", "\003ccc"}))
-- select queries for the 'raw-srv.powerdns.com.' name and SRV type, and answer with a '0 0 65535 srv.powerdns.com.' SRV record, setting the AA bit to 1 and the TTL to 3600s
addAction(AndRule({QNameRule('raw-srv.powerdns.com.'), QTypeRule(DNSQType.SRV)}), SpoofRawAction("\000\000\000\000\255\255\003srv\008powerdns\003com\000", { aa=true, ttl=3600 }))
-- select reverse queries for '127.0.0.1' and answer with 'localhost'
addAction(AndRule({QNameRule('1.0.0.127.in-addr.arpa.'), QTypeRule(DNSQType.PTR)}), SpoofRawAction("\009localhost\000"))
-- rfc8482: Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY via HINFO of value "rfc8482"
addAction(QTypeRule(DNSQType.ANY), SpoofRawAction("\007rfc\056\052\056\050\000", { typeForAny=DNSQType.HINFO }))
```

DNSName:toDNSString() is convenient for converting names to wire format for passing to `SpoofRawAction`.

`sdig dumpluaraw` and `pdnsutil raw-lua-from-content` from PowerDNS can generate raw answers for you:

```
$ pdnsutil raw-lua-from-content SRV '0 0 65535 srv.powerdns.com.'
"\000\000\000\000\255\255\003srv\008powerdns\003com\000"
$ sdig 127.0.0.1 53 open-xchange.com MX recurse dumpluaraw
Reply to question for qname='open-xchange.com.', qtype=MX
Rcode: 0 (No Error), RD: 1, QR: 1, TC: 0, AA: 0, opcode: 0
0 open-xchange.com. IN  MX  "\000c\004mx\049\049\012open\045xchange\003com\000"
0 open-xchange.com. IN  MX  "\000\010\003mx\049\012open\045xchange\003com\000"
0 open-xchange.com. IN  MX  "\000\020\003mx\050\012open\045xchange\003com\000"
```

Parameters

- **rawAnswer** (*string*) – The raw record data
- **rawAnswers** (*{string}*) – A table of raw record data to spoof
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `aa`: bool - Set the AA bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ad`: bool - Set the AD bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ra`: bool - Set the RA bit to this value (true means the bit is set, false means it's cleared). Default is to copy the value of the RD bit from the incoming query.
- `ttl`: int - The TTL of the record.
- `typeForAny`: int - The record type to use when responding to queries of type `ANY`, as using `ANY` for the type of the response record would not make sense.

SpoofSVCAction(*svcParams*[, *options*])

New in version 1.7.0.

Forge a response with the specified SVC record data. If the list contains more than one class:*SVCRecordParameters* (generated via `newSVCRecordParameters()`) object, they are all returned, and should have different priorities. The hints provided in the SVC parameters, if any, will also be added as A/AAAA records in the additional section, using the target name present in the parameters as owner name if it's not empty (root) and the qname instead.

:param list of class:*SVCRecordParameters* svcParams: The record data to return :param table options: A table with key: value pairs with options.

Options:

- `aa`: bool - Set the AA bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ad`: bool - Set the AD bit to this value (true means the bit is set, false means it's cleared). Default is to clear it.
- `ra`: bool - Set the RA bit to this value (true means the bit is set, false means it's cleared). Default is to copy the value of the RD bit from the incoming query.
- `ttl`: int - The TTL of the record.

SpoofPacketAction(*rawPacket*, *len*)

New in version 1.8.0.

Spoof a raw self-generated answer

Parameters

- **rawPacket** (*string*) – The raw wire-ready DNS answer
- **len** (*int*) – The length of the packet

TagAction(*name*, *value*)

Deprecated since version 1.6.0: This function has been deprecated in 1.6.0 and removed in 1.7.0, please use `SetTagAction()` instead.

Associate a tag named `name` with a value of `value` to this query, that will be passed on to the response. Subsequent rules are processed after this action.

Parameters

- **name** (*string*) – The name of the tag to set

- **value** (*string*) – The value of the tag

TagResponseAction (*name*, *value*)

Deprecated since version 1.6.0: This function has been deprecated in 1.6.0 and removed in 1.7.0, please use *SetTagResponseAction()* instead.

Associate a tag named `name` with a value of `value` to this response. Subsequent rules are processed after this action.

Parameters

- **name** (*string*) – The name of the tag to set
- **value** (*string*) – The value of the tag

TCAction ()

Changed in version 1.7.0: This action is now only performed over UDP transports.

Create answer to query with the TC bit set, and the RA bit set to the value of RD in the query, to force the client to TCP. Before 1.7.0 this action was performed even when the query had been received over TCP, which required the use of *TCPRule()* to prevent the TC bit from being set over TCP transports.

TCResponseAction ()

New in version 1.9.0.

Truncate an existing answer, to force the client to TCP. Only applied to answers that will be sent to the client over TCP. In addition to the TC bit being set, all records are removed from the answer, authority and additional sections.

TeeAction (*remote*[, *addECS*[, *local*[, *addProxyProtocol*]]])

Changed in version 1.8.0: Added the optional parameter `local`.

Changed in version 1.9.0: Added the optional parameter `addProxyProtocol`.

Send copy of query to `remote`, keep stats on responses. If `addECS` is set to true, EDNS Client Subnet information will be added to the query. If `addProxyProtocol` is set to true, a Proxy Protocol v2 payload will be prepended in front of the query. The payload will contain the protocol the initial query was received over (UDP or TCP), as well as the initial source and destination addresses and ports. If `local` has provided a value like "192.0.2.53", **dnsdist** will try binding that address as local address when sending the queries. Subsequent rules are processed after this action.

Parameters

- **remote** (*string*) – An IP:PORT combination to send the copied queries to
- **addECS** (*bool*) – Whether to add ECS information. Default false.
- **local** (*str*) – The local address to use to send queries. The default is to let the kernel pick one.
- **addProxyProtocol** (*bool*) – Whether to prepend a proxy protocol v2 payload in front of the query. Default to false.

TempFailureCacheTTLAction (*ttl*)

Deprecated since version 1.6.0.

This function has been deprecated in 1.6.0 and removed in 1.7.0, please use *SetTempFailureCacheTTLAction()* instead.

Set the cache TTL to use for ServFail and Refused replies. TTL is not applied for successful replies. Subsequent rules are processed after this action.

Parameters **ttl** (*int*) – Cache TTL for temporary failure replies

19.2 Configuration Reference

This page lists all configuration options for dnsdist.

Note: When an IPv6 IP:PORT combination is needed, the bracketed syntax from **RFC 3986** should be used. e.g. “[2001:DB8:14::C0FF:FEE]:5300”.

19.2.1 Functions and Types

Within dnsdist several core object types exist:

- *Server*: generated with *newServer()*, represents a downstream server
- *ComboAddress*: represents an IP address and port
- *DNSName*: represents a domain name
- *Netmask*: represents a netmask
- *NetmaskGroup*: represents a group of netmasks
- QPSLimiter: implements a QPS-based filter
- *SuffixMatchNode*: represents a group of domain suffixes for rapid testing of membership
- *DNSHeader*: represents the header of a DNS packet, see *DNSHeader (dh) object*
- *ClientState*: sometimes also called Bind or Frontend, represents the addresses and ports dnsdist is listening on

The existence of most of these objects can mostly be ignored, unless you plan to write your own hooks and policies, but it helps to understand an expressions like:

```
getServer(0).order=12         -- set order of server 0 to 12
getServer(0):addPool("abuse") -- add this server to the abuse pool
```

The **.** means `order` is a data member, while the **:** means `addPool` is a member function.

19.2.2 Global configuration

addCapabilitiesToRetain(*capabilities*)

New in version 1.7.0.

Accept a Linux capability as a string, or a list of these, to retain after startup so that privileged operations can still be performed at runtime. Keeping `CAP_BPF` on kernel 5.8+ for example allows loading eBPF programs and altering eBPF maps at runtime even if the `kernel.unprivileged_bpf_disabled` sysctl is set. Note that this does not grant the capabilities to the process, doing so might be done by running it as root which we don’t advise, or by adding capabilities via the systemd unit file, for example. Please also be aware that switching to a different user via `--uid` will still drop all capabilities.

includeDirectory(*path*)

Include configuration files from `path`.

Parameters **path** (*str*) – The directory to load configuration files from. Each file must end in `.conf`.

reloadAllCertificates()

New in version 1.4.0.

Reload all DNSCrypt and TLS certificates, along with their associated keys.

setSyslogFacility(*facility*)

New in version 1.4.0.

Changed in version 1.6.0: `facility` can now be a string.

Set the syslog logging facility to `facility`.

Parameters or str facility (*int*) – The new facility as a numeric value (raw value as defined in syslog.h), or as a case-insensitive string (“LOCAL0”, or “daemon”, for example). Defaults to LOG_DAEMON.

Listen Sockets

addLocal(*address*[, *options*])

Changed in version 1.4.0: Removed `doTCP` from the options. A listen socket on TCP is always created.

Changed in version 1.5.0: Added `tcpListenQueueSize` parameter.

Changed in version 1.6.0: Added `maxInFlight` and `maxConcurrentTCPConnections` parameters.

Changed in version 1.9.0: Added the `enableProxyProtocol` parameter, which was always `true` before 1.9.0, and the‘‘xskSocket‘‘ one.

Add to the list of listen addresses. Note that for IPv6 link-local addresses, it might be necessary to specify the interface to use: `fe80::1%eth0`. On recent Linux versions specifying the interface via the `interface` parameter should work as well.

Parameters

- **address** (*str*) – The IP Address with an optional port to listen on. The default port is 53.
- **options** (*table*) – A table with key: value pairs with listen options.

Options:

- `doTCP=true`: bool - Also bind on TCP on `address`. Removed in 1.4.0.
- `reusePort=false`: bool - Set the `SO_REUSEPORT` socket option.
- `tcpFastOpenQueueSize=0`: int - Set the TCP Fast Open queue size, enabling TCP Fast Open when available and the value is larger than 0.
- `interface=""`: str - Set the network interface to use.
- `cpus={}`: table - Set the CPU affinity for this listener thread, asking the scheduler to run it on a single CPU id, or a set of CPU ids. This parameter is only available if the OS provides the pthread_setaffinity_np() function.
- `tcpListenQueueSize=SOMAXCONN`: int - Set the size of the listen queue. Default is `SOMAXCONN`.
- `maxInFlight=0`: int - Maximum number of in-flight queries. The default is 0, which disables out-of-order processing.
- `maxConcurrentTCPConnections=0`: int - Maximum number of concurrent incoming TCP connections. The default is 0 which means unlimited.
- `enableProxyProtocol=true`: str - Whether to expect a proxy protocol v2 header in front of incoming queries coming from an address in *setProxyProtocolACL()*. Default is `true`, meaning that queries are expected to have a proxy protocol payload if they come from an address present in the *setProxyProtocolACL()* ACL.
- `xskSocket`: *XskSocket* - A socket to enable `XSK` / `AF_XDP` support for this frontend. See *AF_XDP / XSK* for more information.

```
addLocal('0.0.0.0:5300', { reusePort=true })
```

This will bind to both UDP and TCP on port 5300 with SO_REUSEPORT enabled.

addDOHLocal(*address*[, *certFile(s)*[, *keyFile(s)*[, *urls*[, *options*]]]])

New in version 1.4.0.

Changed in version 1.5.0: `internalPipeBufferSize`, `sendCacheControlHeaders`, `sessionTimeout`, `trustForwardedForHeader` options added. `url` now defaults to `/dns-query` instead of `/`, and does exact matching instead of accepting sub-paths. Added `tcpListenQueueSize` parameter.

Changed in version 1.6.0: `enableRenegotiation`, `exactPathMatching`, `maxConcurrentTCPConnections` and `releaseBuffers` options added. `internalPipeBufferSize` now defaults to 1048576 on Linux.

Changed in version 1.8.0: `certFile` now accepts a *`TLSCertificate`* object or a list of such objects (see *`newTLSCertificate()`*) `additionalAddresses`, `ignoreTLSConfigurationErrors` and `keepIncomingHeaders` options added.

Changed in version 1.9.0: `enableProxyProtocol`, `ktls`, `library`, `proxyProtocolOutsideTLS`, `readAhead`, `tlsAsyncMode` options added.

Listen on the specified address and TCP port for incoming DNS over HTTPS connections, presenting the specified X.509 certificate. See *TLS Certificates Management* for details about the handling of TLS certificates and keys. If no certificate (or key) files are specified, listen for incoming DNS over HTTP connections instead. More information is available in *DNS-over-HTTPS (DoH)*.

Parameters

- **`address`** (*`str`*) – The IP Address with an optional port to listen on. The default port is 443.
- **`certFile(s)`** (*`str`*) – The path to a X.509 certificate file in PEM format, a list of paths to such files, or a *`TLSCertificate`* object.
- **`keyFile(s)`** (*`str`*) – The path to the private key file corresponding to the certificate, or a list of paths to such files, whose order should match the certFile(s) ones. Ignored if `certFile` contains *`TLSCertificate`* objects.
- **`urls`** (*`str-or-list`*) – The path part of a URL, or a list of paths, to accept queries on. Any query with a path matching exactly one of these will be treated as a DoH query (sub-paths can be accepted by setting the `exactPathMatching` to false). The default is /dns-query.
- **`options`** (*`table`*) – A table with key: value pairs with listen options.

Options:

- `reusePort=false`: bool - Set the `SO_REUSEPORT` socket option.
- `tcpFastOpenQueueSize=0`: int - Set the TCP Fast Open queue size, enabling TCP Fast Open when available and the value is larger than 0.
- `interface=""`: str - Set the network interface to use.
- `cpus={}`: table - Set the CPU affinity for this listener thread, asking the scheduler to run it on a single CPU id, or a set of CPU ids. This parameter is only available if the OS provides the pthread_setaffinity_np() function.
- `idleTimeout=30`: int - Set the idle timeout, in seconds.
- `ciphers`: str - The TLS ciphers to use, in OpenSSL format. Ciphers for TLS 1.3 must be specified via `ciphersTLS13`.
- `ciphersTLS13`: str - The TLS ciphers to use for TLS 1.3, in OpenSSL format.
- `serverTokens`: str - The content of the Server: HTTP header returned by dnsdist. The default is “h2o/dnsdist” when `h2o` is used, “nghttp2-<version>/dnsdist” when `nghttp2` is.
- `customResponseHeaders={}`: table - Set custom HTTP header(s) returned by dnsdist.
- `ocspResponses`: list - List of files containing OCSP responses, in the same order than the certificates and keys, that will be used to provide OCSP stapling responses.

- minTLSVersion: str - Minimum version of the TLS protocol to support. Possible values are 'tls1.0', 'tls1.1', 'tls1.2' and 'tls1.3'. Default is to require at least TLS 1.0.
- numberOfTicketsKeys: int - The maximum number of tickets keys to keep in memory at the same time. Only one key is marked as active and used to encrypt new tickets while the remaining ones can still be used to decrypt existing tickets after a rotation. Default to 5.
- ticketKeyFile: str - The path to a file from where TLS tickets keys should be loaded, to support **RFC 5077**. These keys should be rotated often and never written to persistent storage to preserve forward secrecy. The default is to generate a random key. dnsdist supports several tickets keys to be able to decrypt existing sessions after the rotation. See *TLS Sessions Management* for more information.
- ticketsKeysRotationDelay: int - Set the delay before the TLS tickets key is rotated, in seconds. Default is 43200 (12h). A value of 0 disables the automatic rotation, which might be useful when ticketKeyFile is used.
- sessionTimeout: int - Set the TLS session lifetime in seconds, this is used both for TLS ticket lifetime and for sessions kept in memory.
- sessionTickets: bool - Whether session resumption via session tickets is enabled. Default is true, meaning tickets are enabled.
- numberOfStoredSessions: int - The maximum number of sessions kept in memory at the same time. Default is 20480. Setting this value to 0 disables stored session entirely.
- preferServerCiphers: bool - Whether to prefer the order of ciphers set by the server instead of the one set by the client. Default is true, meaning that the order of the server is used. For OpenSSL >= 1.1.1, setting this option also enables the temporary re-prioritization of the ChaCha20-Poly1305 cipher if the client prioritizes it.
- keyLogFile: str - Write the TLS keys in the specified file so that an external program can decrypt TLS exchanges, in the format described in https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format. Note that this feature requires OpenSSL >= 1.1.1.
- sendCacheControlHeaders: bool - Whether to parse the response to find the lowest TTL and set a HTTP Cache-Control header accordingly. Default is true.
- trustForwardedForHeader: bool - Whether to parse any existing X-Forwarded-For header in the HTTP query and use the right-most value as the client source address and port, for ACL checks, rules, logging and so on. Default is false.
- tcpListenQueueSize=SOMAXCONN: int - Set the size of the listen queue. Default is SOMAXCONN.
- internalPipeBufferSize=0: int - Set the size in bytes of the internal buffer of the pipes used internally to pass queries and responses between threads. Requires support for F_SETPIPE_SZ which is present in Linux since 2.6.35. The actual size might be rounded up to a multiple of a page size. 0 means that the OS default size is used. The default value is 0, except on Linux where it is 1048576 since 1.6.0.
- exactPathMatching=true: bool - Whether to do exact path matching of the query path against the paths configured in urls (true, the default since 1.5.0) or to accepts sub-paths (false, and was the default before 1.5.0). This option was introduced in 1.6.0.
- maxConcurrentTCPConnections=0: int - Maximum number of concurrent incoming TCP connections. The default is 0 which means unlimited.
- releaseBuffers=true: bool - Whether OpenSSL should release its I/O buffers when a connection goes idle, saving roughly 35 kB of memory per connection.
- enableRenegotiation=false: bool - Whether secure TLS renegotiation should be enabled. Disabled by default since it increases the attack surface and is seldom used for DNS.
- keepIncomingHeaders: bool - Whether to retain the incoming headers in memory, to be able to use *HTTPHeaderRule()* or DNSQuestion.getHTTPHeaders(). Default is false. Before 1.8.0 the headers were always kept in-memory.

- `additionalAddresses`: list - List of additional addresses (with port) to listen on. Using this option instead of creating a new frontend for each address avoids the creation of new thread and Frontend objects, reducing the memory usage. The drawback is that there will be a single set of metrics for all addresses.
- `ignoreTLSConfigurationErrors=false`: bool - Ignore TLS configuration errors (such as invalid certificate path) and just issue a warning instead of aborting the whole process
- `library`: str - Which underlying HTTP2 library should be used, either h2o or nghttp2. Until 1.9.0 only h2o was available, but the use of this library is now deprecated as it is no longer maintained. nghttp2 is the new default since 1.9.0.
- `ktls=false`: bool - Whether to enable the experimental kernel TLS support on Linux, if both the kernel and the OpenSSL library support it. Default is false.
- `tlsAsyncMode=false`: bool - Whether to enable experimental asynchronous TLS I/O operations if the `nghttp2` library is used, `OpenSSL` is used as the TLS implementation and an asynchronous capable SSL engine (or provider) is loaded. See also *`loadTLSEngine()`* or *`loadTLSProvider()`* to load the engine (or provider).
- `readAhead`: bool - When the TLS provider is set to OpenSSL, whether we tell the library to read as many input bytes as possible, which leads to better performance by reducing the number of syscalls. Default is true.
- `proxyProtocolOutsideTLS`: bool - When the use of incoming proxy protocol is enabled, whether the payload is prepended after the start of the TLS session (so inside, meaning it is protected by the TLS layer providing encryption and authentication) or not (outside, meaning it is in clear-text). Default is false which means inside. Note that most third-party software like HAproxy expect the proxy protocol payload to be outside, in clear-text.
- `enableProxyProtocol=true`: bool - Whether to expect a proxy protocol v2 header in front of incoming queries coming from an address in *`setProxyProtocolACL()`*. Default is `true`, meaning that queries are expected to have a proxy protocol payload if they come from an address present in the *`setProxyProtocolACL()`* ACL.

addDOH3Local (*address*, *certFile(s)*, *keyFile(s)*[, *options*])

New in version 1.9.0.

Listen on the specified address and UDP port for incoming DNS over HTTP3 connections, presenting the specified X.509 certificate. See *TLS Certificates Management* for details about the handling of TLS certificates and keys. More information is available in *DNS-over-HTTP/3 (DoH3)*.

Parameters

- **address** (*str*) – The IP Address with an optional port to listen on. The default port is 443.
- **certFile(s)** (*str*) – The path to a X.509 certificate file in PEM format, a list of paths to such files, or a *`TLSCertificate`* object.
- **keyFile(s)** (*str*) – The path to the private key file corresponding to the certificate, or a list of paths to such files, whose order should match the certFile(s) ones. Ignored if `certFile` contains *`TLSCertificate`* objects.
- **options** (*table*) – A table with key: value pairs with listen options.

Options:

- `reusePort=false`: bool - Set the `SO_REUSEPORT` socket option.
- `interface=""`: str - Set the network interface to use.
- `cpus={}`: table - Set the CPU affinity for this listener thread, asking the scheduler to run it on a single CPU id, or a set of CPU ids. This parameter is only available if the OS provides the pthread_setaffinity_np() function.
- `idleTimeout=5`: int - Set the idle timeout, in seconds.

- `internalPipeBufferSize=0`: int - Set the size in bytes of the internal buffer of the pipes used internally to pass queries and responses between threads. Requires support for `F_SETPIPE_SZ` which is present in Linux since 2.6.35. The actual size might be rounded up to a multiple of a page size. 0 means that the OS default size is used. The default value is 0, except on Linux where it is 1048576 since 1.6.0.
- `maxInFlight=65535`: int - Maximum number of in-flight queries. The default is 0, which disables out-of-order processing.
- `congestionControlAlgo="reno"`: str - The congestion control algorithm to be chosen between `reno`, `cubic` and `bbr`.
- `keyLogFile`: str - Write the TLS keys in the specified file so that an external program can decrypt TLS exchanges, in the format described in https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format.

addDOQLocal (*address*, *certFile(s)*, *keyFile(s)*[, *options*])

New in version 1.9.0.

Listen on the specified address and UDP port for incoming DNS over QUIC connections, presenting the specified X.509 certificate. See *TLS Certificates Management* for details about the handling of TLS certificates and keys. More information is available at *DNS-over-QUIC (DoQ)*.

Parameters

- **address** (*str*) – The IP Address with an optional port to listen on. The default port is 853.
- **certFile(s)** (*str*) – The path to a X.509 certificate file in PEM format, a list of paths to such files, or a *TLSCertificate* object.
- **keyFile(s)** (*str*) – The path to the private key file corresponding to the certificate, or a list of paths to such files, whose order should match the certFile(s) ones. Ignored if `certFile` contains *TLSCertificate* objects.
- **options** (*table*) – A table with key: value pairs with listen options.

Options:

- `reusePort=false`: bool - Set the `SO_REUSEPORT` socket option.
- `interface=""`: str - Set the network interface to use.
- `cpus={}`: table - Set the CPU affinity for this listener thread, asking the scheduler to run it on a single CPU id, or a set of CPU ids. This parameter is only available if the OS provides the pthread_setaffinity_np() function.
- `idleTimeout=5`: int - Set the idle timeout, in seconds.
- `internalPipeBufferSize=0`: int - Set the size in bytes of the internal buffer of the pipes used internally to pass queries and responses between threads. Requires support for `F_SETPIPE_SZ` which is present in Linux since 2.6.35. The actual size might be rounded up to a multiple of a page size. 0 means that the OS default size is used. The default value is 0, except on Linux where it is 1048576 since 1.6.0.
- `maxInFlight=65535`: int - Maximum number of in-flight queries. The default is 0, which disables out-of-order processing.
- `congestionControlAlgo="reno"`: str - The congestion control algorithm to be chosen between `reno`, `cubic` and `bbr`.
- `keyLogFile`: str - Write the TLS keys in the specified file so that an external program can decrypt TLS exchanges, in the format described in https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format.

addTLSLocal (*address*, *certFile(s)*, *keyFile(s)*[, *options*])

Changed in version 1.4.0: `ciphersTLS13`, `minTLSVersion`, `ocspResponses`, `preferServerCiphers`, `keyLogFile` options added.

Changed in version 1.5.0: `sessionTimeout` and `tcpListenQueueSize` options added.

Changed in version 1.6.0: `enableRenegotiation`, `maxConcurrentTCPConnections`, `maxInFlight` and `releaseBuffers` options added.

Changed in version 1.8.0: `tlsAsyncMode` option added.

Changed in version 1.8.0: `certFile` now accepts a *`TLSCertificate`* object or a list of such objects (see *`newTLSCertificate()`*). `additionalAddresses`, `ignoreTLSConfigurationErrors` and `ktls` options added.

Changed in version 1.9.0: `enableProxyProtocol`, `readAhead` and `proxyProtocolOutsideTLS` options added.

Listen on the specified address and TCP port for incoming DNS over TLS connections, presenting the specified X.509 certificate. See *TLS Certificates Management* for details about the handling of TLS certificates and keys. More information is available at *DNS-over-TLS*.

Parameters

- **`address`** (*`str`*) – The IP Address with an optional port to listen on. The default port is 853.
- **`certFile(s)`** (*`str`*) – The path to a X.509 certificate file in PEM format, a list of paths to such files, or a *`TLSCertificate`* object.
- **`keyFile(s)`** (*`str`*) – The path to the private key file corresponding to the certificate, or a list of paths to such files, whose order should match the certFile(s) ones. Ignored if `certFile` contains *`TLSCertificate`* objects.
- **`options`** (*`table`*) – A table with key: value pairs with listen options.

Options:

- `reusePort=false`: bool - Set the `SO_REUSEPORT` socket option.
- `tcpFastOpenQueueSize=0`: int - Set the TCP Fast Open queue size, enabling TCP Fast Open when available and the value is larger than 0.
- `interface=""`: str - Set the network interface to use.
- `cpus={}`: table - Set the CPU affinity for this listener thread, asking the scheduler to run it on a single CPU id, or a set of CPU ids. This parameter is only available if the OS provides the pthread_setaffinity_np() function.
- `provider`: str - The TLS library to use between GnuTLS and OpenSSL, if they were available and enabled at compilation time. Default is to use OpenSSL when available.
- `ciphers`: str - The TLS ciphers to use. The exact format depends on the provider used. When the OpenSSL provider is used, ciphers for TLS 1.3 must be specified via `ciphersTLS13`.
- `ciphersTLS13`: str - The ciphers to use for TLS 1.3, when the OpenSSL provider is used. When the GnuTLS provider is used, `ciphers` applies regardless of the TLS protocol and this setting is not used.
- `numberOfTicketsKeys`: int - The maximum number of tickets keys to keep in memory at the same time, if the provider supports it (GnuTLS doesn't, OpenSSL does). Only one key is marked as active and used to encrypt new tickets while the remaining ones can still be used to decrypt existing tickets after a rotation. Default to 5.
- `ticketKeyFile`: str - The path to a file from where TLS tickets keys should be loaded, to support **RFC 5077**. These keys should be rotated often and never written to persistent storage to preserve forward secrecy. The default is to generate a random key. The OpenSSL provider supports several tickets keys to be able to decrypt existing sessions after the rotation, while the GnuTLS provider only supports one key. See *TLS Sessions Management* for more information.
- `ticketsKeysRotationDelay`: int - Set the delay before the TLS tickets key is rotated, in seconds. Default is 43200 (12h). A value of 0 disables the automatic rotation, which might be useful when `ticketKeyFile` is used.

- `sessionTimeout`: int - Set the TLS session lifetime in seconds, this is used both for TLS ticket lifetime and for sessions kept in memory.
- `sessionTickets`: bool - Whether session resumption via session tickets is enabled. Default is true, meaning tickets are enabled.
- `numberOfStoredSessions`: int - The maximum number of sessions kept in memory at the same time. At this time this is only supported by the OpenSSL provider, as stored sessions are not supported with the GnuTLS one. Default is 20480. Setting this value to 0 disables stored session entirely.
- `ocspResponses`: list - List of files containing OCSP responses, in the same order than the certificates and keys, that will be used to provide OCSP stapling responses.
- `minTLSVersion`: str - Minimum version of the TLS protocol to support. Possible values are 'tls1.0', 'tls1.1', 'tls1.2' and 'tls1.3'. Default is to require at least TLS 1.0. Note that this value is ignored when the GnuTLS provider is in use, and the `ciphers` option should be set accordingly instead. For example, 'NORMAL:!VERS-TLS1.0:!VERS-TLS1.1' will disable TLS 1.0 and 1.1.
- `preferServerCiphers`: bool - Whether to prefer the order of ciphers set by the server instead of the one set by the client. Default is true, meaning that the order of the server is used. For OpenSSL >= 1.1.1, setting this option also enables the temporary re-prioritization of the ChaCha20-Poly1305 cipher if the client prioritizes it.
- `keyLogFile`: str - Write the TLS keys in the specified file so that an external program can decrypt TLS exchanges, in the format described in https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format. Note that this feature requires OpenSSL >= 1.1.1.
- `tcpListenQueueSize=SOMAXCONN`: int - Set the size of the listen queue. Default is `SOMAXCONN`.
- `maxInFlight=0`: int - Maximum number of in-flight queries. The default is 0, which disables out-of-order processing.
- `maxConcurrentTCPConnections=0`: int - Maximum number of concurrent incoming TCP connections. The default is 0 which means unlimited.
- `releaseBuffers=true`: bool - Whether OpenSSL should release its I/O buffers when a connection goes idle, saving roughly 35 kB of memory per connection.
- `enableRenegotiation=false`: bool - Whether secure TLS renegotiation should be enabled (OpenSSL only, the GnuTLS provider does not support it). Disabled by default since it increases the attack surface and is seldom used for DNS.
- `tlsAsyncMode=false`: bool - Whether to enable experimental asynchronous TLS I/O operations if OpenSSL is used as the TLS implementation and an asynchronous capable SSL engine (or provider) is loaded. See also *`loadTLSEngine()`* or *`loadTLSProvider()`* to load the engine (or provider).
- `additionalAddresses`: list - List of additional addresses (with port) to listen on. Using this option instead of creating a new frontend for each address avoids the creation of new thread and Frontend objects, reducing the memory usage. The drawback is that there will be a single set of metrics for all addresses.
- `ignoreTLSConfigurationErrors=false`: bool - Ignore TLS configuration errors (such as invalid certificate path) and just issue a warning instead of aborting the whole process
- `ktls=false`: bool - Whether to enable the experimental kernel TLS support on Linux, if both the kernel and the OpenSSL library support it. Default is false.
- `readAhead`: bool - When the TLS provider is set to OpenSSL, whether we tell the library to read as many input bytes as possible, which leads to better performance by reducing the number of syscalls. Default is true.
- `proxyProtocolOutsideTLS`: bool - When the use of incoming proxy protocol is enabled, whether the payload is prepended after the start of the TLS session (so inside, meaning it is protected by the TLS layer providing encryption and authentication) or not (outside, meaning it is in clear-text). Default is false which means inside. Note that most third-party software like HAproxy expect the proxy protocol payload to be outside, in clear-text.

- `enableProxyProtocol=true`: str - Whether to expect a proxy protocol v2 header in front of incoming queries coming from an address in *`setProxyProtocolACL()`*. Default is `true`, meaning that queries are expected to have a proxy protocol payload if they come from an address present in the *`setProxyProtocolACL()`* ACL.

`setLocal`(*address*[, *options*])

Remove the list of listen addresses and add a new one.

Parameters

- **address** (*str*) – The IP Address with an optional port to listen on. The default port is 53.
- **options** (*table*) – A table with key: value pairs with listen options.

The options that can be set are the same as *`addLocal()`*.

Control Socket, Console and Webserver

`addConsoleACL`(*netmask*)

Add a netmask to the existing console ACL, allowing remote clients to connect to the console. Please make sure that encryption has been enabled with *`setKey()`* before doing so. The default is to only allow 127.0.0.1/8 and ::1/128.

Parameters **netmask** (*str*) – A CIDR netmask, e.g. `"192.0.2.0/24"`. Without a subnetmask, only the specific address is allowed.

`clearConsoleHistory`()

New in version 1.6.0.

Clear the internal (in-memory) buffers of console commands. These buffers are used to provide the *`delta()`* command and console completion and history, and can end up being quite large when a lot of commands are issued via the console, consuming a noticeable amount of memory.

`controlSocket`(*address*)

Bind to `addr` and listen for a connection for the console. Since 1.3.0 only connections from local users are allowed by default, *`addConsoleACL()`* and *`setConsoleACL()`* can be used to enable remote connections. Please make sure that encryption has been enabled with *`setKey()`* before doing so. Enabling encryption is also strongly advised for local connections, since not enabling it allows any local user to connect to the console.

Parameters **address** (*str*) – An IP address with optional port. By default, the port is 5199.

`delta`()

Issuing *delta* on the console will print the changes to the configuration that have been made since startup.

`inClientStartup`()

Returns true while the console client is parsing the configuration.

`inConfigCheck`()

New in version 1.5.0.

Returns true while the configuration is being checked, ie when run with `--check-config`.

`makeKey`()

Generate and print an encryption key.

`setConsoleConnectionsLogging`(*enabled*)

Whether to log the opening and closing of console connections.

Parameters **enabled** (*bool*) – Default to true.

`setConsoleMaximumConcurrentConnections`(*max*)

New in version 1.6.0.

Set the maximum number of concurrent console connections.

Parameters **max** (*int*) – The maximum number of concurrent console connections, or 0 which means an unlimited number. Defaults to 100

setKey (*key*)

Use `key` as shared secret between the client and the server

Parameters **key** (*str*) – An encoded key, as generated by *makeKey()*

setConsoleACL (*netmasks*)

Remove the existing console ACL and add the netmasks from the table, allowing remote clients to connect to the console. Please make sure that encryption has been enabled with *setKey()* before doing so.

Parameters **netmasks** (*{str}*) – A table of CIDR netmask, e.g. `{"192.0.2.0/24", "2001:DB8:14::/56"}`. Without a subnetmask, only the specific address is allowed.

showConsoleACL ()

Print a list of all netmasks allowed to connect to the console.

testCrypto ()

Test the crypto code, will report errors when something is not ok.

setConsoleOutputMaxMsgSize (*size*)

Set the maximum size in bytes of a single console message, default set to 10 MB.

Parameters **size** (*int*) – The new maximum size.

Webserver configuration

hashPassword (*password*[, *workFactor*])

New in version 1.7.0.

Hash the supplied password using a random salt, and returns a string that can be used with *setWebserverConfig()*.

Parameters

- **– password** (*string*) – The password to hash
- **– workFactor** (*int*) – The work factor to use for the hash function (currently scrypt), as a power of two. Default is 1024.

webserver (*listen_address*[, *password*[, *apikey*[, *customHeaders*[, *acl*]]]])

Changed in version 1.5.0: `acl` optional parameter added.

Changed in version 1.6.0: The `password` parameter is now optional. The use of optional parameters is now deprecated. Please use *setWebserverConfig()* instead.

Changed in version 1.8.0: The `password`, `apikey`, `customHeaders` and `acl` parameters is no longer supported. Please use *setWebserverConfig()* instead.

Launch the *Built-in webserver* with statistics and the API. Note that the parameters are global, so the parameter from the last `webserver` will override any existing ones. For this reason *setWebserverConfig()* should be used instead of specifying optional parameters here.

Parameters

- **listen_address** (*str*) – The IP address and Port to listen on
- **password** (*str*) – The password required to access the webserver
- **apikey** (*str*) – The key required to access the API
- **customHeaders** (*{[str]=str,...}*) – Allows setting custom headers and removing the defaults
- **acl** (*str*) – List of netmasks, as a string, that are allowed to open a connection to the web server. Defaults to "127.0.0.1, ::1". It accepts the same syntax that *NetmaskGroup:addMask()* does

setAPIWritable(*allow*[, *dir*])

Allow modifications via the API. Optionally saving these changes to disk. Modifications done via the API will not be written to the configuration by default and will not persist after a reload

Parameters

- **allow** (*bool*) – Set to true to allow modification through the API
- **dir** (*str*) – A valid directory where the configuration files will be written by the API.

setWebserverConfig(*options*)

Changed in version 1.5.0: `acl` optional parameter added.

Changed in version 1.6.0: `statsRequireAuthentication`, `maxConcurrentConnections` optional parameters added.

Changed in version 1.7.0: The optional `password` and `apiKey` parameters now accept hashed passwords. The optional `hashPlaintextCredentials` parameter has been added.

Changed in version 1.8.0: `apiRequiresAuthentication`, `dashboardRequiresAuthentication` optional parameters added.

Setup webserver configuration. See *webserver()*.

Parameters **options** (*table*) – A table with key: value pairs with webserver options.

Options:

- `password=newPassword`: string - Set the password used to access the internal webserver. Since 1.7.0 the password should be hashed and salted via the *hashPassword()* command.
- `apiKey=newKey`: string - Changes the API Key (set to an empty string do disable it). Since 1.7.0 the key should be hashed and salted via the *hashPassword()* command.
- `customHeaders={[str]=str,...}`: map of string - Allows setting custom headers and removing the defaults.
- `acl=newACL`: string - List of IP addresses, as a string, that are allowed to open a connection to the web server. Defaults to "127.0.0.1, ::1".
- `apiRequiresAuthentication`: bool - Whether access to the API (/api endpoints) require a valid API key. Defaults to true.
- `dashboardRequiresAuthentication`: bool - Whether access to the internal dashboard requires a valid password. Defaults to true.
- `statsRequireAuthentication`: bool - Whether access to the statistics (/metrics and /jsonstat endpoints) require a valid password or API key. Defaults to true.
- `maxConcurrentConnections`: int - The maximum number of concurrent web connections, or 0 which means an unlimited number. Defaults to 100.
- `hashPlaintextCredentials`: bool - Whether passwords and API keys provided in plaintext should be hashed during startup, to prevent the plaintext versions from staying in memory. Doing so increases significantly the cost of verifying credentials. Defaults to false.

registerWebHandler(*path*, *handler*)

Register a function named `handler` that will be called for every query sent to the exact `path` path. The function will receive a *WebRequest* object and a *WebResponse* object, representing respectively the HTTP request received and the HTTP response to send. For example a handler registered for '/foo' will receive these queries:

- `GET /foo`
- `POST /foo`
- `GET /foo?param=1`

But not queries for /foobar or /foo/bar.

A sample handler function could be:

```
function customHTTPHandler(req, resp)
  local get = req.getvars
  local headers = req.headers

  if req.path ~= '/foo' or req.version ~= 11 or req.method ~= 'GET' or get[
↪'param'] ~= '42' or headers['custom'] ~= 'foobar' then
    resp.status = 500
    return
  end

  resp.status = 200
  resp.body = 'It works!'
  resp.headers = { ['Foo']='Bar'}
end

registerWebHandler('/foo', customHTTPHandler)
```

Parameters

- **`path`** (*`str`*) – Path to register the handler for.
- **`handler`** (*`function`*) – The Lua function to register.

`showWebserverConfig`()

New in version 1.7.0.

Show the current webserver configuration. See *`webserver()`*.

Access Control Lists

`addACL`(*netmask*)

Add a netmask to the existing ACL controlling which clients can send UDP, TCP, DNS over TLS and DNS over HTTPS queries. See *Access Control* for more information.

Parameters **`netmask`** (*`str`*) – A CIDR netmask, e.g. `"192.0.2.0/24"`. Without a subnetmask, only the specific address is allowed.

`rmACL`(*netmask*)

Remove a network from the existing ACL controlling which clients can send UDP, TCP, DNS over TLS and DNS over HTTPS queries. See *Access Control* for more information. This function only removes previously added entries, it does not remove subnets of entries.

Parameters **`netmask`** (*`str`*) – A CIDR netmask, e.g. `"192.0.2.0/24"`. Without a subnetmask, only the specific address is allowed.

```
addACL("192.0.2.0/24") -- for example add subnet to the ACL
rmACL("192.0.2.10")    -- does NOT work, the ACL is unchanged
rmACL("192.0.2.0/24")  -- does work, the exact match is removed from the ACL
```

`setACL`(*netmasks*)

Remove the existing ACL and add the netmasks from the table of those allowed to send UDP, TCP, DNS over TLS and DNS over HTTPS queries. See *Access Control* for more information.

Parameters **`netmasks`** (*`{str}`*) – A table of CIDR netmask, e.g. `{"192.0.2.0/24", "2001:DB8:14::/56"}`. Without a subnetmask, only the specific address is allowed.

`setACLFromFile`(*fname*)

New in version 1.6.0.

Reset the ACL to the list of netmasks from the given file. See *Access Control* for more information.

Parameters **`fname`** (*`str`*) – The path to a file containing a list of netmasks. Empty lines or lines starting with “#” are ignored.

setProxyProtocolACL(*netmasks*)

New in version 1.6.0.

Set the list of netmasks from which a Proxy Protocol header will be required, over UDP, TCP and DNS over TLS. The default is empty. Note that a proxy protocol payload will be required from these clients, regular DNS queries will no longer be accepted if they are not preceded by a proxy protocol payload. Be also aware that, if *setProxyProtocolApplyACLToProxiedClients()* is set (default is false), the general ACL will be applied to the source IP address as seen by dnsdist first, but also to the source IP address provided in the Proxy Protocol header.

Parameters **netmasks** (*{str}*) – A table of CIDR netmask, e.g. `{"192.0.2.0/24", "2001:DB8:14::/56"}`. Without a subnetmask, only the specific address is allowed.

setProxyProtocolApplyACLToProxiedClients(*apply*)

New in version 1.6.0.

Whether the general ACL should be applied to the source IP address provided in the Proxy Protocol header, in addition to being applied to the source IP address as seen by dnsdist first.

Parameters **apply** (*bool*) – Whether it should be applied or not (default is false).

showACL()

Print a list of all netmasks allowed to send queries over UDP, TCP, DNS over TLS and DNS over HTTPS. See *Access Control* for more information.

EDNS Client Subnet

setECSOverride(*bool*)

When `useClientSubnet` in *newServer()* is set and dnsdist adds an EDNS Client Subnet Client option to the query, override an existing option already present in the query, if any. Note that it's not recommended to enable `setECSOverride` in front of an authoritative server responding with EDNS Client Subnet information as mismatching data (ECS scopes) can confuse clients and lead to SERVFAIL responses on downstream nameservers.

Parameters **bool** – Whether to override an existing EDNS Client Subnet option present in the query. Defaults to false

setECSSourcePrefixV4(*prefix*)

When `useClientSubnet` in *newServer()* is set and dnsdist adds an EDNS Client Subnet Client option to the query, truncate the requestors IPv4 address to `prefix` bits

Parameters **prefix** (*int*) – The prefix length

setECSSourcePrefixV6(*prefix*)

When `useClientSubnet` in *newServer()* is set and dnsdist adds an EDNS Client Subnet Client option to the query, truncate the requestor's IPv6 address to bits

Parameters **prefix** (*int*) – The prefix length

Ringbuffers

setRingBuffersLockRetries(*num*)

Deprecated since version 1.8.0: Deprecated in 1.8.0 in favor of *setRingBuffersOptions()* which provides more options.

Set the number of shards to attempt to lock without blocking before giving up and simply blocking while waiting for the next shard to be available

Parameters **num** (*int*) – The maximum number of attempts. Defaults to 5 if there is more than one shard, 0 otherwise.

setRingBuffersOptions(*options*)

New in version 1.8.0.

Set the rings buffers configuration

Parameters **options** (*table*) – A table with key: value pairs with options.

Options:

- `lockRetries`: int - Set the number of shards to attempt to lock without blocking before giving up and simply blocking while waiting for the next shard to be available. Default to 5 if there is more than one shard, 0 otherwise
- `recordQueries`: boolean - Whether to record queries in the ring buffers. Default is true. Note that `grepq()`, several top* commands (`topClients()`, `topQueries()`, . . .) and the *Dynamic Blocks* require this to be enabled.
- `recordResponses`: boolean - Whether to record responses in the ring buffers. Default is true. Note that `grepq()`, several top* commands (`topResponses()`, `topSlow()`, . . .) and the *Dynamic Blocks* require this to be enabled.

setRingBuffersSize(*num*[, *numberOfShards*])

Changed in version 1.6.0: `numberOfShards` defaults to 10.

Set the capacity of the ringbuffers used for live traffic inspection to `num`, and the number of shards to `numberOfShards` if specified. Increasing the number of entries comes at both a memory cost (around 250 MB for 1 million entries) and a CPU processing cost, so we strongly advise not going over 1 million entries.

Parameters

- **num** (*int*) – The maximum amount of queries to keep in the ringbuffer. Defaults to 10000
- **numberOfShards** (*int*) – the number of shards to use to limit lock contention. Default is 10, used to be 1 before 1.6.0

19.2.3 Servers

newServer(*server_string*)

newServer(*server_table*)

Changed in version 1.4.0: Added `checkInterval`, `checkTimeout` and `rise` to server_table.

Changed in version 1.5.0: Added `useProxyProtocol` to server_table.

Changed in version 1.6.0: Added `maxInFlight` to server_table.

Changed in version 1.7.0: Added `addXForwardedHeaders`, `caStore`, `checkTCP`, `ciphers`, `ciphers13`, `dohPath`, `enableRenegotiation`, `releaseBuffers`, `subjectName`, `tcpOnly`, `tls` and `validateCertificates` to server_table.

Changed in version 1.8.0: Added `autoUpgrade`, `autoUpgradeDoHKey`, `autoUpgradeInterval`, `autoUpgradeKeep`, `autoUpgradePool`, `maxConcurrentTCPConnections`, `subjectAddr`, `lazyHealthCheckSampleSize`, `lazyHealthCheckMinSampleCount`, `lazyHealthCheckThreshold`, `lazyHealthCheckFailedInterval`, `lazyHealthCheckMode`, `lazyHealthCheckUseExponentialBackOff`, `lazyHealthCheckMaxBackOff`, `lazyHealthCheckWhenUpgraded`, `healthCheckMode` and `ktls` to server_table.

Changed in version 1.9.0: Added `MACAddr`, `proxyProtocolAdvertiseTLS` and `xskSockets` to server_table.

Parameters

- **server_string** (*str*) – A simple IP:PORT string.
- **server_table** (*table*) – A table with at least an `address` key

Add a new backend server. Call this function with either a string:

```
newServer(
  "IP:PORT" -- IP and PORT of the backend server
)
```

or a table:

```
newServer({ ... })
```

where the elements in the table can be:

Keyword	Type	Description
address	ip:port	ip and port of the backend server (mandatory)
id	string	Use a pre-defined UUID instead of a random one
qps	number	Limit the number of queries per second to number, when using the *firstAvailable* policy
order	number	The order of this server, used by the *leastOutstanding* and *firstAvailable* policies
weight	number	The weight of this server, used by the *wrandom*, *whashed* and *chashed* policies, default: 1. Supported values are a minimum of 1, and a maximum of 2147483647.
pool	string|{string}	The pools this server belongs to (unset or empty string means default pool) as a string or table of strings
retries	number	The number of TCP connection attempts to the backend, for a given query
tcpConnectTimeout	number	The timeout (in seconds) of a TCP connection attempt
tcpSendTimeout	number	The timeout (in seconds) of a TCP write attempt
tcpRecvTimeout	number	The timeout (in seconds) of a TCP read attempt
tcpFastOpen	bool	Whether to enable TCP Fast Open
ipBindAddrNoPort	bool	Whether to enable IP_BIND_ADDRESS_NO_PORT if available, default: true
name	string	The name associated to this backend, for display purpose
checkClass	number	Use number as QCLASS in the health-check query, default: DNSClass.IN
checkName	string	Use string as QNAME in the health-check query, default: "a.root-servers.net."

Continued on next page

Table 1 – continued from previous page

Keyword	Type	Description
`checkType`	`string`	Use `string` as QTYPE in the health-check query, default: `"A"`
`checkFunction`	`function`	Use this function to dynamically set the QNAME, QTYPE and QCLASS to use in the health-check query (see *Healthcheck*)
`checkTimeout`	`number`	The timeout (in milliseconds) of a health-check query, default: 1000 (1s)
`setCD`	`bool`	Set the CD (Checking Disabled) flag in the health-check query, default: false
`maxCheckFailures`	`number`	Allow `number` check failures before declaring the backend down, default: 1
`checkInterval`	`number`	The time in seconds between health checks
`mustResolve`	`bool`	Set to true when the health check MUST return a RCODE different from NXDomain, ServFail and Refused. Default is false, meaning that every RCODE except ServFail is considered valid
`useClientSubnet`	`bool`	Add the client's IP address in the EDNS Client Subnet option when forwarding the query to this backend
`source`	`string`	**The source address or interface to use for queries to** The following formats are supported: • address, e.g. `"192.0.2.2"` • interface name, e.g. `"eth0"` • address@interface, e.g. `"192.0.2.2@eth0"`
`addXPF`	`number`	Add the client's IP address and port to the query, along with the original destination address and port, using the experimental XPF record from draft-bellis-dnsop-xpf and the specified option code. Default is disabled (0). This is a deprecated feature that will be removed in the near future.

Continued on next page

Table 1 – continued from previous page

Keyword	Type	Description
sockets	number	Number of UDP sockets (and thus source ports) used toward the backend server, defaults to a single one. Note that for backends which are multithreaded, this setting will have an effect on the number of cores that will be used to process traffic from dnsdist. For example you may want to set 'sockets' to a number somewhat higher than the number of worker threads configured in the backend, particularly if the Linux kernel is being used to distribute traffic to multiple threads listening on the same socket (via *reuseport*). See also `setRandomizedOutgoingSockets()`.
disableZeroScope	bool	Disable the EDNS Client Subnet 'zero scope' feature, which does a cache lookup for an answer valid for all subnets (ECS scope of 0) before adding ECS information to the query and doing the regular lookup. This requires the `parseECS` option of the corresponding cache to be set to true
rise	number	Require `number` consecutive successful checks before declaring the backend up, default: 1
useProxyProtocol	bool	Add a proxy protocol header to the query, passing along the client's IP address and port along with the original destination address and port. Default is disabled.
reconnectOnUp	bool	Close and reopen the sockets when a server transits from Down to Up. This helps when an interface is missing when dnsdist is started. Default is disabled.
maxInFlight	number	Maximum number of in-flight queries. The default is 0, which disables out-of-order processing. It should only be enabled if the backend does support out-of-order processing. As of 1.6.0, out-of-order processing needs to be enabled on the frontend as well, via `addLocal()` and/or `addTLSLocal()`. Note that out-of-order is always enabled on DoH frontends.

Continued on next page

Table 1 – continued from previous page

Keyword	Type	Description
tcpOnly	bool	Always forward queries to that backend over TCP, never over UDP. Always enabled for TLS backends. Default is false.
checkTCP	bool	Whether to do healthcheck queries over TCP, instead of UDP. Always enabled for DNS over TLS backend. Default is false.
tls	string	Enable DNS over TLS communications for this backend, or DNS over HTTPS if dohPath is set, using the TLS provider ("openssl" or "gnutls") passed in parameter. Default is an empty string, which means this backend is used for plain UDP and TCP.
caStore	string	Specifies the path to the CA certificate file, in PEM format, to use to check the certificate presented by the backend. Default is an empty string, which means to use the system CA store. Note that this directive is only used if validateCertificates is set.
ciphers	string	The TLS ciphers to use. The exact format depends on the provider used. When the OpenSSL provider is used, ciphers for TLS 1.3 must be specified via ciphersTLS13.
ciphersTLS13	string	The ciphers to use for TLS 1.3, when the OpenSSL provider is used. When the GnuTLS provider is used, ciphers applies regardless of the TLS protocol and this setting is not used.
subjectName	string	The subject name passed in the SNI value of the TLS handshake, and against which to validate the certificate presented by the backend. Default is empty. If set this value supersedes any subjectAddr one.
subjectAddr	string	The subject IP address passed in the SNI value of the TLS handshake, and against which to validate the certificate presented by the backend. Default is empty.
validateCertificates	bool	Whether the certificate presented by the backend should be validated against the CA store (see caStore). Default is true.

Continued on next page

Table 1 – continued from previous page

Keyword	Type	Description
dohPath	string	Enable DNS over HTTPS communication for this backend, using POST queries to the HTTP host supplied as subjectName and the HTTP path supplied in this parameter.
addXForwardedHeaders	bool	Whether to add X-Forwarded-For, X-Forwarded-Port and X-Forwarded-Proto headers to a DNS over HTTPS backend.
releaseBuffers	bool	Whether OpenSSL should release its I/O buffers when a connection goes idle, saving roughly 35 kB of memory per connection. Default to true.
enableRenegotiation	bool	Whether secure TLS renegotiation should be enabled. Disabled by default since it increases the attack surface and is seldom used for DNS.
autoUpgrade	bool	Whether to use the 'Discovery of Designated Resolvers' mechanism to automatically upgrade a Do53 backend to DoT or DoH, depending on the priorities present in the SVCB record returned by the backend. Default to false.
autoUpgradeInterval	number	If autoUpgrade is set, how often to check if an upgrade is available, in seconds. Default is 3600 seconds.
autoUpgradeKeep	bool	If autoUpgrade is set, whether to keep the existing Do53 backend around after an upgrade. Default is false which means the Do53 backend will be replaced by the upgraded one.
autoUpgradePool	string	If autoUpgrade is set, in which pool to place the newly upgraded backend. Default is empty which means the backend is placed in the default pool.
autoUpgradeDoHKey	number	If autoUpgrade is set, the value to use for the SVC key corresponding to the DoH path. Default is 7.
maxConcurrentTCPConnections	number	Maximum number of TCP connections to that backend. When that limit is reached, queries routed to that backend that cannot be forwarded over an existing connection will be dropped. Default is 0 which means no limit.

Continued on next page

Table 1 – continued from previous page

Keyword	Type	Description
`healthCheckMode`	`string`	The health-check mode to use: ‘auto’ which sends health-check queries every `checkInterval` seconds, ‘up’ which considers that the backend is always available, ‘down’ that it is always not available, and ‘lazy’ which only sends health-check queries after a configurable amount of regular queries have failed (see `lazyHealthCheckSampleSize`, `lazyHealthCheckMinSampleCount`, `lazyHealthCheckThreshold`, `lazyHealthCheckFailedInterval` and `lazyHealthCheckMode` for more information). Default is ‘auto’. See *Healthcheck* for a more detailed explanation.
`lazyHealthCheckFailedInterval`	`number`	The interval, in seconds, between health-check queries in ‘lazy’ mode. Note that when `lazyHealthCheckUseExponentialBackOff` is set to true, the interval doubles between every queries. These queries are only sent when a threshold of failing regular queries has been reached, and until the backend is available again. Default is 30 seconds.
`lazyHealthCheckMinSampleCount`	`number`	The minimum amount of regular queries that should have been recorded before the `lazyHealthCheckThreshold` threshold can be applied. Default is 1 which means only one query is needed.
`lazyHealthCheckMode`	`string`	The ‘lazy’ health-check mode: ‘TimeoutOnly’ means that only timeout and I/O errors of regular queries will be considered for the `lazyHealthCheckThreshold`, while ‘TimeoutOrServFail’ will also consider ‘Server Failure’ answers. Default is ‘TimeoutOrServFail’.

Continued on next page

Table 1 – continued from previous page

Keyword	Type	Description
`lazyHealthCheckSampleSize`	`number`	The maximum size of the sample of queries to record and consider for the `lazyHealthCheckThreshold`. Default is 100, which means the result (failure or success) of the last 100 queries will be considered.
`lazyHealthCheckThreshold`	`number`	The threshold, as a percentage, of queries that should fail for the 'lazy' health-check to be triggered when `healthCheckMode` is set to `lazy`. The default is 20 which means 20% of the last `lazyHealthCheckSampleSize` queries should fail for a health-check to be triggered.
`lazyHealthCheckUseExponentialBackOff`	`bool`	Whether the 'lazy' health-check should use an exponential back-off instead of a fixed value, between health-check probes. The default is false which means that after a backend has been moved to the 'down' state health-check probes are sent every `lazyHealthCheckFailedInterval` seconds. When set to true, the delay between each probe starts at `lazyHealthCheckFailedInterval` seconds and double between every probe, capped at `lazyHealthCheckMaxBackOff` seconds.
`lazyHealthCheckMaxBackOff`	`number`	This value, in seconds, caps the time between two health-check queries when `lazyHealthCheckUseExponentialBackOff` is set to true. The default is 3600 which means that at most one hour will pass between two health-check queries.
`lazyHealthCheckWhenUpgraded`	`bool`	Whether the auto-upgraded version of this backend (see `autoUpgrade`) should use the lazy health-checking mode. Default is false, which means it will use the regular health-checking mode.

Continued on next page

Table 1 – continued from previous page

Keyword	Type	Description
`ktls`	`bool`	Whether to enable the experimental kernel TLS support on Linux, if both the kernel and the OpenSSL library support it. Default is false. Currently both DoT and DoH backend support this option.
`proxyProtocolAdvertiseTLS`	`bool`	Whether to set the SSL Proxy Protocol TLV in the proxy protocol payload sent to the backend if the query was received over an encrypted channel (DNSCrypt, DoQ, DoH or DoT). Requires `useProxyProtocol=true`. Default is false.
`xskSockets`	`array`	An array of `XskSocket` objects to enable `XSK` / `AF_XDP` support for this backend. See *AF_XDP / XSK* for more information.
`MACAddr`	`str`	When the `xskSocket` option is set, this parameter can be used to specify the destination MAC address to use to reach the backend. If this options is not specified, dnsdist will try to get it from the IP of the backend by looking into the system's MAC address table, but it will fail if the corresponding MAC address is not present.

`getServer`(*index*) → Server

Changed in version 1.5.0: `index` might be an UUID.

Get a `Server`

Parameters **`or str index`** (*`int`*) – The number of the server (as seen in `showServers()`) or its UUID as a string.

Returns The `Server` object or nil

`getServers`()

Returns a table with all defined servers.

`rmServer`(*index*)
`rmServer`(*uuid*)
`rmServer`(*server*)

Changed in version 1.5.0: `uuid` selection added.

Remove a backend server.

Parameters

- **`or str index`** (*`int`*) – The number of the server (as seen in `showServers()`), its UUID as a string, or a server object.
- **`server`** (`Server`) – A `Server` object as returned by e.g. `getServer()`.

Server Functions

A server object returned by *getServer()* can be manipulated with these functions.

class Server

This object represents a backend server. It has several methods.

:addPool(*pool*)

Add this server to a pool.

Parameters **pool** (*str*) – The pool to add the server to

:getLatency() → double

New in version 1.6.0.

Return the average latency of this server over the last 128 UDP queries, in microseconds.

Returns The number of outstanding queries

:getName() → string

Get the name of this server.

Returns The name of the server, or an empty string if it does not have one

:getNameWithAddr() → string

Get the name plus IP address and port of the server

Returns A string containing the server name if any plus the server address and port

:getDrops() → int

New in version 1.6.0.

Get the number of dropped queries for this server.

Returns The number of dropped queries

:getOutstanding() → int

Get the number of outstanding queries for this server.

Returns The number of outstanding queries

:isUp() → bool

Returns the up status of the server

Returns true when the server is up, false otherwise

:rmPool(*pool*)

Removes the server from the named pool

Parameters **pool** (*str*) – The pool to remove the server from

:setAuto([*status*])

Set the server in the default auto state. This will enable health check queries that will set the server `up` and `down` appropriately.

Parameters **status** (*bool*) – Set the initial status of the server to `up` (true) or `down` (false) instead of using the last known status

:setDown()

Set the server in a `DOWN` state. The server will not receive queries and the health checks are disabled.

:setLazyAuto([*status*])

New in version 1.8.0.

Set the server in the 'lazy' health-check mode. This will enable health check queries, but only after a configurable threshold of failing regular queries has been reached and only for a short time. See *Healthcheck* for a more detailed explanation.

Parameters **status** (*bool*) – Set the initial status of the server to `up` (true) or `down` (false) instead of using the last known status

:setQPS(*limit*)

Limit the queries per second for this server.

Parameters **limit** (*int*) – The maximum number of queries per second

:setUp()

Set the server in an `UP` state. This server will still receive queries and health checks are disabled

Apart from the functions, a *Server* object has these attributes:

name

The name of the server

upStatus

Whether or not this server is up or down

order

The order of the server

weight

The weight of the server

19.2.4 Pools

*Server*s can be part of any number of pools. Pools are automatically created when a server is added to a pool (with *newServer()*), or can be manually created with *getPool()*. Servers that are not assigned to a specific pool get assigned to the default pool that is always present, identified by the empty string `''`.

getPool(*name*) → ServerPool

Returns a *ServerPool*. If the pool does not exist yet, it is created.

Parameters **name** (*string*) – The name of the pool

getPoolServers(*name*) → [Server]

Returns a list of *Server*s or nil.

Parameters **name** (*string*) – The name of the pool

getPoolNames() → [table of names]

New in version 1.8.0.

Returns a table of all pool names

showPools()

Display the name, associated cache, server policy and associated servers for every pool.

class ServerPool

This represents the pool where zero or more servers are part of.

:getCache() → PacketCache

Returns the *PacketCache* for this pool or nil.

:getECS()

Whether dnsdist will add EDNS Client Subnet information to the query before looking up into the cache, when all servers from this pool are down. For more information see *ServerPool:setECS()*.

:setCache(*cache*)

Adds `cache` as the pool's cache.

Parameters **cache** (PacketCache) – The new cache to add to the pool

:unsetCache()

Removes the cache from this pool.

:setECS()

Set to true if dnsdist should add EDNS Client Subnet information to the query before looking up into the cache, when all servers from this pool are down. If at least one server is up, the preference of the selected server is used, this parameter is only useful if all the backends in this pool are down and have EDNS Client Subnet enabled, since the queries in the cache will have been inserted with ECS information. Default is false.

PacketCache

A Pool can have a packet cache to answer queries directly instead of going to the backend. See *Caching Responses* for a how to.

newPacketCache (*maxEntries[, maxTTL=86400[, minTTL=0[, temporaryFailureTTL=60[, staleTTL=60[, dontAge=false[, numberOfShards=1[, deferrableInsertLock=true[, maxNegativeTTL=3600[, parseECS=false]]]]]]]]]*) → PacketCache

Deprecated since version 1.4.0.

Creates a new `PacketCache` with the settings specified.

Parameters

- **maxEntries** (*int*) – The maximum number of entries in this cache
- **maxTTL** (*int*) – Cap the TTL for records to his number
- **minTTL** (*int*) – Don't cache entries with a TTL lower than this
- **temporaryFailureTTL** (*int*) – On a SERVFAIL or REFUSED from the backend, cache for this amount of seconds
- **staleTTL** (*int*) – When the backend servers are not reachable, and global configuration `setStaleCacheEntriesTTL` is set appropriately, TTL that will be used when a stale cache entry is returned
- **dontAge** (*bool*) – Don't reduce TTLs when serving from the cache. Use this when **dnsdist** fronts a cluster of authoritative servers
- **numberOfShards** (*int*) – Number of shards to divide the cache into, to reduce lock contention
- **deferrableInsertLock** (*bool*) – Whether the cache should give up insertion if the lock is held by another thread, or simply wait to get the lock
- **maxNegativeTTL** (*int*) – Cache a NXDomain or NoData answer from the backend for at most this amount of seconds, even if the TTL of the SOA record is higher
- **parseECS** (*bool*) – Whether any EDNS Client Subnet option present in the query should be extracted and stored to be able to detect hash collisions involving queries with the same qname, qtype and qclass but a different incoming ECS value. Enabling this option adds a parsing cost and only makes sense if at least one backend might send different responses based on the ECS value, so it's disabled by default

newPacketCache (*maxEntries*[, *options*]) → PacketCache

New in version 1.4.0.

Changed in version 1.6.0: `cookieHashing` parameter added. `numberOfShards` now defaults to 20.

Changed in version 1.7.0: `skipOptions` parameter added.

Changed in version 1.9.0: `maximumEntrySize` parameter added.

Creates a new `PacketCache` with the settings specified.

Parameters **maxEntries** (*int*) – The maximum number of entries in this cache

Options:

- `deferrableInsertLock=true`: bool - Whether the cache should give up insertion if the lock is held by another thread, or simply wait to get the lock.
- `dontAge=false`: bool - Don't reduce TTLs when serving from the cache. Use this when **dnsdist** fronts a cluster of authoritative servers.

- `keepStaleData=false`: bool - Whether to suspend the removal of expired entries from the cache when there is no backend available in at least one of the pools using this cache.
- `maxNegativeTTL=3600`: int - Cache a NXDomain or NoData answer from the backend for at most this amount of seconds, even if the TTL of the SOA record is higher.
- `maxTTL=86400`: int - Cap the TTL for records to his number.
- `minTTL=0`: int - Don't cache entries with a TTL lower than this.
- `numberOfShards=20`: int - Number of shards to divide the cache into, to reduce lock contention. Used to be 1 (no shards) before 1.6.0, and is now 20.
- `parseECS=false`: bool - Whether any EDNS Client Subnet option present in the query should be extracted and stored to be able to detect hash collisions involving queries with the same qname, qtype and qclass but a different incoming ECS value. Enabling this option adds a parsing cost and only makes sense if at least one backend might send different responses based on the ECS value, so it's disabled by default. Enabling this option is required for the 'zero scope' option to work
- `staleTTL=60`: int - When the backend servers are not reachable, and global configuration `setStaleCacheEntriesTTL` is set appropriately, TTL that will be used when a stale cache entry is returned.
- `temporaryFailureTTL=60`: int - On a SERVFAIL or REFUSED from the backend, cache for this amount of seconds..
- `cookieHashing=false`: bool - If true, EDNS Cookie values will be hashed, resulting in separate entries for different cookies in the packet cache. This is required if the backend is sending answers with EDNS Cookies, otherwise a client might receive an answer with the wrong cookie.
- `skipOptions={}`: Extra list of EDNS option codes to skip when hashing the packet (if `cookieHashing` above is false, EDNS cookie option number will be added to this list internally).
- `maximumEntrySize=4096`: int - The maximum size, in bytes, of a DNS packet that can be inserted into the packet cache. Default is 4096 bytes, which was the fixed size before 1.9.0, and is also a hard limit for UDP responses.

class PacketCache

Represents a cache that can be part of *`ServerPool`*.

:dump (*fname*)

Dump a summary of the cache entries to a file.

Parameters **fname** (*str*) – The path to a file where the cache summary should be dumped. Note that if the target file already exists, it will not be overwritten.

:expunge (*n*)

Remove entries from the cache, leaving at most `n` entries

Parameters **n** (*int*) – Number of entries to keep

:expungeByName (*name*[, *qtype=DNSQType.ANY*[, *suffixMatch=false*]])

Changed in version 1.6.0: `name` can now also be a string

Remove entries matching `name` and type from the cache.

Parameters

- **name** (`DNSName`) – The name to expunge
- **qtype** (*int*) – The type to expunge, can be a pre-defined *DNSQType*
- **suffixMatch** (*bool*) – When set to true, remove all entries under `name`

:getAddressListByDomain (*domain*)

New in version 1.8.0.

This method looks up the answers present in the cache for the supplied domain, and returns the list of addresses present in the answer section of these answers (in A records for IPv4 addresses, and AAAA records for IPv6 ones). The addresses are returned as a list of *`ComboAddress`* objects.

Parameters **domain** (DNSName) – The domain to look for

:getDomainListByAddress(*addr*)
New in version 1.8.0.

Return a list of domains, as *DNSName* objects, for which an answer is present in the cache and has a corresponding A record (for IPv4 addresses) or AAAA record (for IPv6 addresses) in the answer section.

Parameters **addr** (ComboAddress) – The address to look for

:getStats()
New in version 1.4.0.

Return the cache stats (number of entries, hits, misses, deferred lookups, deferred inserts, lookup collisions, insert collisions and TTL too shorts) as a Lua table.

:isFull() → bool
Return true if the cache has reached the maximum number of entries.

:printStats()
Print the cache stats (number of entries, hits, misses, deferred lookups, deferred inserts, lookup collisions, insert collisions and TTL too shorts).

:purgeExpired(*n*)
Remove expired entries from the cache until there is at most `n` entries remaining in the cache.

Parameters **n** (*int*) – Number of entries to keep

:toString() → string
Return the number of entries in the Packet Cache, and the maximum number of entries

19.2.5 Client State

Also called frontend or bind, the Client State object returned by *getBind()* and listed with *showBinds()* represents an address and port dnsdist is listening on.

getBind(*index*) → ClientState
Return a *ClientState* object.

Parameters **index** (*int*) – The object index

getBindCount()
New in version 1.5.0.

Return the number of binds (Do53, DNSCrypt, DoH and DoT).

ClientState functions

class ClientState
This object represents an address and port dnsdist is listening on. When `reuseport` is in use, several ClientState objects can be present for the same address and port.

:attachFilter(*filter*)
Attach a BPF filter to this frontend.

Parameters **filter** (BPFFilter) – The filter to attach to this frontend

:detachFilter()
Remove the BPF filter associated to this frontend, if any.

:getEffectiveTLSProvider() → string
New in version 1.7.0.

Return the name of the TLS provider actually used.

:getRequestedTLSProvider() → string
New in version 1.7.0.

Return the name of the TLS provider requested in the configuration.

:getType() → string
New in version 1.7.0.

Return the type of the frontend: UDP, UDP (DNSCrypt), TCP, TCP (DNSCrypt), TCP (DNS over TLS) or TCP (DNS over HTTPS).

:toString() → string
Return the address and port this frontend is listening on.

Returns The address and port this frontend is listening on

muted
If set to true, queries received on this frontend will be normally processed and sent to a backend if needed, but no response will be ever be sent to the client over UDP. TCP queries are processed normally and responses sent to the client.

19.2.6 Status, Statistics and More

dumpStats()
Print all statistics dnsdist gathers

getDOHFrontend(*idx*)
New in version 1.4.0.

Return the `DOHFrontend` object for the DNS over HTTPS bind of index `idx`.

getDOHFrontendCount()
New in version 1.5.0.

Return the number of `DOHFrontend` binds.

getDOH3Frontend(*idx*)
New in version 1.9.0.

Return the `DOH3Frontend` object for the DNS over HTTP3 bind of index `idx`.

getDOH3FrontendCount()
New in version 1.9.0.

Return the number of `DOH3Frontend` binds.

getDOQFrontend(*idx*)
New in version 1.9.0.

Return the `DOQFrontend` object for the DNS over QUIC bind of index `idx`.

getDOQFrontendCount()
New in version 1.9.0.

Return the number of `DOQFrontend` binds.

getListOfAddressesOfNetworkInterface(*itf*)
New in version 1.8.0.

Return the list of addresses configured on a given network interface, as strings. This function requires support for `getifaddrs`, which is known to be present on FreeBSD, Linux, and OpenBSD at least.

Parameters **itf** (*str*) – The name of the network interface

getListOfNetworkInterfaces()
New in version 1.8.0.

Return the list of network interfaces configured on the system, as strings. This function requires support for `getifaddrs`, which is known to be present on FreeBSD, Linux, and OpenBSD at least.

getListOfRangesOfNetworkInterface(*itf*)
New in version 1.8.0.

Return the list of network ranges configured on a given network interface, as strings. This function requires support for `getifaddrs`, which is known to be present on FreeBSD, Linux, and OpenBSD at least.

Parameters **itf** (`str`) – The name of the network interface

getMACAddress(*ip*) → str
New in version 1.8.0.

Return the link-level address (MAC) corresponding to the supplied neighbour IP address, if known by the kernel. The link-level address is returned as a raw binary string. An empty string is returned if no matching entry has been found. This function is only implemented on Linux.

Parameters **ip** (`str`) – The IP address, IPv4 or IPv6, to look up the corresponding link-level address for.

getOutgoingTLSSessionCacheSize()
New in version 1.7.0.

Return the number of TLS sessions (for outgoing connections) currently cached.

getTLSContext(*idx*)
Return the TLSContext object for the context of index `idx`.

getTLSFrontend(*idx*)
Return the TLSFrontend object for the TLS bind of index `idx`.

getTLSFrontendCount()
New in version 1.5.0.

Return the number of TLSFrontend binds.

getTopCacheHitResponseRules([*top*])
New in version 1.6.0.

Return the cache-hit response rules that matched the most.

Parameters **top** (`int`) – How many response rules to return. Default is 10.

getTopCacheInsertedResponseRules([*top*])
New in version 1.8.0.

Return the cache-inserted response rules that matched the most.

Parameters **top** (`int`) – How many response rules to return. Default is 10.

getTopResponseRules([*top*])
New in version 1.6.0.

Return the response rules that matched the most.

Parameters **top** (`int`) – How many response rules to return. Default is 10.

getTopRules([*top*])
New in version 1.6.0.

Return the rules that matched the most.

Parameters **top** (`int`) – How many rules to return. Default is 10.

getTopSelfAnsweredRules([*top*])
New in version 1.6.0.

Return the self-answered rules that matched the most.

Parameters **top** (`int`) – How many rules to return. Default is 10.

grepq(*selector*[, *num*[, *options*]])

grepq(*selectors*[, *num*[, *options*]])

Changed in version 1.9.0: `options` optional parameter table added.

Prints the last `num` queries and responses matching `selector` or `selectors`. Queries and responses are accounted in separate ring buffers, and answers from the packet cache are not stored in the response ring buffer. Therefore, the `num` queries and `num` responses in the output may not always match up.

The selector can be:

- a netmask (e.g. '192.0.2.0/24')
- a DNS name (e.g. 'dnsdist.org')
- a response time (e.g. '100ms')

Parameters

- **selector** (*str*) – Select queries based on this property.
- **selectors** (*{str}*) – A lua table of selectors. Only queries matching all selectors are shown
- **num** (*int*) – Show a maximum of `num` recent queries+responses.
- **options** (*table*) – A table with key: value pairs with options described below.

Options:

- `outputFile=path`: string - Write the output of the command to the supplied file, instead of the standard output.

setStructuredLogging(*enable*[, *options*])

New in version 1.9.0.

Set whether log messages should be in a structured-logging-like format. This is turned off by default. The resulting format looks like this (when timestamps are enabled via `--log-timestamps` and with `levelPrefix="prio"` and `timeFormat="ISO8601"`):

```
ts="2023-11-06T12:04:58+0100" prio="Info" msg="Added downstream server 127.0.0.
↪1:53"
```

And with `levelPrefix="level"` and `timeFormat="numeric"`):

```
ts="1699268815.133" level="Info" msg="Added downstream server 127.0.0.1:53"
```

Parameters

- **enable** (*bool*) – Set to true if you want to enable structured logging
- **options** (*table*) – A table with key: value pairs with options described below.

Options:

- `levelPrefix=prefix`: string - Set the prefix for the log level. Default is `prio`.
- `timeFormat=format`: string - Set the time format. Supported values are `ISO8601` and `numeric`. Default is `numeric`.

setVerbose(*verbose*)

New in version 1.8.0.

Set whether log messages issued at the verbose level should be logged. This is turned off by default.

Parameters **verbose** (*bool*) – Set to true if you want to enable verbose logging

getVerbose()

New in version 1.8.0.

Get whether log messages issued at the verbose level should be logged. This is turned off by default.

setVerboseHealthChecks(*verbose*)

Set whether health check errors should be logged. This is turned off by default.

Parameters **verbose** (`bool`) – Set to true if you want to enable health check errors logging

setVerboseLogDestination(*dest*)

New in version 1.8.0.

Set a destination file to write the 'verbose' log messages to, instead of sending them to syslog and/or the standard output which is the default. Note that these messages will no longer be sent to syslog or the standard output once this option has been set. There is no rotation or file size limit. Only use this feature for debugging under active operator control.

Parameters **dest** (`str`) – The destination file

showBinds()

Print a list of all the current addresses and ports dnsdist is listening on, also called `frontends`

showDOHFrontends()

New in version 1.4.0.

Print the list of all available DNS over HTTPS frontends.

showDOH3Frontends()

New in version 1.9.0.

Print the list of all available DNS over HTTP/3 frontends.

showDOHResponseCodes()

New in version 1.4.0.

Print the HTTP response codes statistics for all available DNS over HTTPS frontends.

showDOQFrontends()

New in version 1.9.0.

Print the list of all available DNS over QUIC frontends.

showResponseLatency()

Show a plot of the response time latency distribution

showServers([*options*])

Changed in version 1.4.0: `options` optional parameter added

This function shows all backend servers currently configured and some statistics. These statics have the following fields:

- `#` - The number of the server, can be used as the argument for `getServer()`
- `UUID` - The UUID of the backend. Can be set with the `id` option of `newServer()`
- `Address` - The IP address and port of the server
- `State` - The current state of the server
- `Qps` - Current number of queries per second
- `Qlim` - Configured maximum number of queries per second
- `Ord` - The order number of the server
- `Wt` - The weight of the server
- `Queries` - Total amount of queries sent to this server
- `Drops` - Number of queries that were dropped by this server
- `Drate` - Number of queries dropped per second by this server
- `Lat` - The latency of this server in milliseconds
- `Pools` - The pools this server belongs to

Parameters **options** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.

showTCPStats()

Show some statistics regarding TCP

showTLSContexts()

Print the list of all available DNS over TLS contexts.

showTLSErrorCounters()

New in version 1.4.0.

Display metrics about TLS handshake failures.

showVersion()

Print the version of dnsdist

topBandwidth([*num*])

Print the top `num` clients that consume the most bandwidth.

Parameters **num** (*int*) – Number to show, defaults to 10.

topCacheHitResponseRules([*top*[, *options*]])

New in version 1.6.0.

This function shows the cache-hit response rules that matched the most.

Parameters

- **top** (*int*) – How many rules to show.
- **options** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.

topCacheInsertedResponseRules([*top*[, *options*]])

New in version 1.8.0.

This function shows the cache-inserted response rules that matched the most.

Parameters

- **top** (*int*) – How many rules to show.
- **options** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.

topClients([*num*])

Print the top `num` clients sending the most queries over length of ringbuffer

Parameters **num** (*int*) – Number to show, defaults to 10.

topQueries([*num*[, *labels*]])

Print the `num` most popular QNAMEs from queries. Optionally grouped by the rightmost `labels` DNS labels.

Parameters

- **num** (*int*) – Number to show, defaults to 10
- **label** (*int*) – Number of labels to cut down to

topResponses([*num*[, *rcode*[, *labels*]]])

Print the `num` most seen responses with an RCODE of `rcode`. Optionally grouped by the rightmost `labels` DNS labels.

Parameters

- **num** (*int*) – Number to show, defaults to 10
- **rcode** (*int*) – *Response code*, defaults to 0 (No Error)
- **label** (*int*) – Number of labels to cut down to

topResponseRules([*top*[, *options*]])

New in version 1.6.0.

This function shows the response rules that matched the most.

Parameters

- **top** (*int*) – How many rules to show.
- **options** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.

topRules([*top*[, *options*]])

New in version 1.6.0.

This function shows the rules that matched the most.

Parameters

- **top** (*int*) – How many rules to show.
- **options** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.

topSelfAnsweredResponseRules([*top*[, *options*]])

New in version 1.6.0.

This function shows the self-answered response rules that matched the most.

Parameters

- **top** (*int*) – How many rules to show.
- **options** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.

topSlow([*num*[, *limit*[, *labels*]]])

Print the `num` slowest queries that are slower than `limit` milliseconds. Optionally grouped by the rightmost `labels` DNS labels.

Parameters

- **num** (*int*) – Number to show, defaults to 10
- **limit** (*int*) – Show queries slower than this amount of milliseconds, defaults to 2000
- **label** (*int*) – Number of labels to cut down to

19.2.7 Dynamic Blocks

addDynamicBlock(*address*, *message*[, *action*[, *seconds*[, *clientIPMask*[, *clientIPPortMask*]]]])

New in version 1.9.0.

Manually block an IP address or range with `message` for (optionally) a number of seconds. The default number of seconds to block for is 10.

Parameters

- **address** – A *ComboAddress* or string representing an IPv4 or IPv6 address
- **message** (*string*) – The message to show next to the blocks
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to DNSAction.None, meaning the one set with *setDynBlocksAction()* is used)
- **seconds** (*int*) – The number of seconds this block to expire
- **clientIPMask** (*int*) – The network mask to apply to the address. Default is 32 for IPv4, 128 for IPv6.
- **clientIPPortMask** (*int*) – The port mask to use to specify a range of ports to match, when the clients are behind a CG-NAT.

Please see the documentation for *setDynBlocksAction()* to confirm which actions are supported by the action paramater.

addDynBlocks(*addresses*, *message*[, *seconds=10*[, *action*]])

Block a set of addresses with `message` for (optionally) a number of seconds. The default number of seconds to block for is 10. Since 1.3.0, the use of a *DynBlockRulesGroup* is a much more efficient way of doing the same thing.

Parameters

- **addresses** – set of Addresses as returned by an exceed function
- **message** (*string*) – The message to show next to the blocks
- **seconds** (*int*) – The number of seconds this block to expire
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to DNSAction.None, meaning the one set with *setDynBlocksAction()* is used)

Please see the documentation for *setDynBlocksAction()* to confirm which actions are supported by the action paramater.

clearDynBlocks()

Remove all current dynamic blocks.

getDynamicBlocks()

New in version 1.9.0.

Return an associative table of active network-based dynamic blocks. The keys are the network IP or range that are blocked, the value are *DynBlock* objects.

getDynamicBlocksSMT()

New in version 1.9.0.

Return an associative table of active domain-based (Suffix Match Tree or SMT) dynamic blocks. The keys are the domains that are blocked, the values are *DynBlock* objects.

showDynBlocks()

List all dynamic blocks in effect.

setDynBlocksAction(*action*)

Set which action is performed when a query is blocked. Only DNSAction.Drop (the default), DNSAction.NoOp, DNSAction.NXDomain, DNSAction.Refused, DNSAction.Truncate and DNSAction.NoRecurse are supported.

setDynBlocksPurgeInterval(*sec*)

New in version 1.6.0.

Set at which interval, in seconds, the expired dynamic blocks entries will be effectively removed from the tree. Entries are not applied anymore as soon as they expire, but they remain in the tree for a while for performance reasons. Removing them makes the addition of new entries faster and frees up the memory they use. Setting this value to 0 disable the purging mechanism, so entries will remain in the tree.

Parameters **`sec`** (*`int`*) – The interval between two runs of the cleaning algorithm, in seconds. Default is 60 (1 minute), 0 means disabled.

`class DynBlock`

New in version 1.9.0.

Represent the current state of a dynamic block.

`action`

The action of this block, as an integer representing a *DNSAction*.

`blocks`

The number of queries blocked.

`bpf`

Whether this block is using eBPF, as a boolean.

`domain`

The domain that is blocked, as a string, for Suffix Match Tree blocks.

`reason`

The reason why this block was inserted, as a string.

`until`

The time (in seconds since Epoch) at which the block will expire.

`warning`

Whether this block is only a warning one (true) or is really enforced (false).

Getting addresses that exceeded parameters

`exceedServFails`(*rate*, *seconds*)

Get set of addresses that exceed `rate` servfails/s over `seconds` seconds

Parameters

- **`rate`** (*`int`*) – Number of Servfails per second to exceed
- **`seconds`** (*`int`*) – Number of seconds the rate has been exceeded

`exceedNXDOMAINs`(*rate*, *seconds*)

get set of addresses that exceed `rate` NXDOMAIN/s over `seconds` seconds

Parameters

- **`rate`** (*`int`*) – Number of NXDOMAIN per second to exceed
- **`seconds`** (*`int`*) – Number of seconds the rate has been exceeded

`exceedRespByterate`(*rate*, *seconds*)

get set of addresses that exceeded `rate` bytes/s answers over `seconds` seconds

Parameters

- **`rate`** (*`int`*) – Number of bytes per second to exceed
- **`seconds`** (*`int`*) – Number of seconds the rate has been exceeded

`exceedQRate`(*rate*, *seconds*)

Get set of address that exceed `rate` queries/s over `seconds` seconds

Parameters

- **`rate`** (*`int`*) – Number of queries per second to exceed
- **`seconds`** (*`int`*) – Number of seconds the rate has been exceeded

exceedQTypeRate (*type*, *rate*, *seconds*)

Get set of address that exceed `rate` queries/s for queries of QType `type` over `seconds` seconds

Parameters

- **type** (*int*) – QType
- **rate** (*int*) – Number of QType queries per second to exceed
- **seconds** (*int*) – Number of seconds the rate has been exceeded

DynBlockRulesGroup

Instead of using several *exceed*()* lines, dnsdist 1.3.0 introduced a new *DynBlockRulesGroup* object which can be used to group dynamic block rules.

See *Dynamic Rule Generation* for more information about the case where using a *DynBlockRulesGroup* might be faster than the existing rules.

dynBlockRulesGroup() → DynBlockRulesGroup

Creates a new `DynBlockRulesGroup` object.

class DynBlockRulesGroup

Represents a group of dynamic block rules.

:setCacheMissRatio(*ratio*, *seconds*, *reason*, *blockingTime*, *minimumNumberOfResponses*, *minimumGlobalCacheHitRatio*[, *action*[, *warningRate*]])

New in version 1.9.0.

Adds a rate-limiting rule for the ratio of cache-misses responses over the total number of responses for a given client. A minimum global cache-hit ratio has to specified to prevent false-positive when the cache is empty.

Parameters

- **ratio** (*float*) – Ratio of cache-miss responses per second over the total number of responses for this client to exceed
- **seconds** (*int*) – Number of seconds the ratio has been exceeded
- **reason** (*string*) – The message to show next to the blocks
- **blockingTime** (*int*) – The number of seconds this block to expire
- **minimumNumberOfResponses** (*int*) – How many total responses is required for this rule to apply
- **minimumGlobalCacheHitRatio** (*float*) – The minimum global cache-hit ratio (over all pools, so `cache-hits / (cache-hits + cache-misses)`) for that rule to be applied.
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to the one set with `setDynBlocksAction()`)
- **warningRatio** (*float*) – If set to a non-zero value, the ratio above which a warning message will be issued and a no-op block inserted

:setMasks(*v4*, *v6*, *port*)

New in version 1.7.0.

Set the number of bits to keep in the IP address when inserting a block. The default is 32 for IPv4 and 128 for IPv6, meaning that only the exact address is blocked, but in some scenarios it might make sense to block a whole /64 IPv6 range instead of a single address, for example. It is also possible to take the IPv4 UDP and TCP ports into account, for CGNAT deployments, by setting the number of bits of the port to consider. For example passing 2 as the last parameter, which only makes sense if the previous parameters are respectively 32 and 128, will split a given IP address into four port ranges: 0-16383, 16384-32767, 32768-49151 and 49152-65535.

Parameters

- **v4** (*int*) – Number of bits to keep for IPv4 addresses. Default is 32
- **v6** (*int*) – Number of bits to keep for IPv6 addresses. Default is 128
- **port** (*int*) – Number of bits of port to consider over IPv4. Default is 0 meaning that the port is not taken into account

:setQueryRate (*rate*, *seconds*, *reason*, *blockingTime*[, *action*[, *warningRate*]])

Adds a query rate-limiting rule, equivalent to: ` addDynBlocks(exceedQRate(rate, seconds), reason, blockingTime, action) `

Parameters

- **rate** (*int*) – Number of queries per second to exceed
- **seconds** (*int*) – Number of seconds the rate has been exceeded
- **reason** (*string*) – The message to show next to the blocks
- **blockingTime** (*int*) – The number of seconds this block to expire
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to the one set with `setDynBlocksAction()`)
- **warningRate** (*int*) – If set to a non-zero value, the rate above which a warning message will be issued and a no-op block inserted

:setNewBlockInsertedHook (*hook*)

New in version 1.9.0.

Set a Lua function that will be called everytime a new dynamic block is inserted. The function receives:

- an integer whose value is 0 if the block is Netmask-based one (Client IP or range) and 1 instead (Domain name suffix)
- the key (Client IP/range or domain suffix) as a string
- the reason of the block as a string
- the action of the block as an integer
- the duration of the block in seconds
- whether this is a warning block (true) or not (false)

:setRCodeRate (*rcode*, *rate*, *seconds*, *reason*, *blockingTime*[, *action*[, *warningRate*]])

> **Note:** Cache hits are inserted into the in-memory ring buffers since 1.8.0, so they are now considered when computing the rcode rate.

Adds a rate-limiting rule for responses of code `rcode`, equivalent to: ` addDynBlocks(exceedServfails(rcode, rate, seconds), reason, blockingTime, action) `

Parameters

- **rcode** (*int*) – The response code
- **rate** (*int*) – Number of responses per second to exceed
- **seconds** (*int*) – Number of seconds the rate has been exceeded
- **reason** (*string*) – The message to show next to the blocks
- **blockingTime** (*int*) – The number of seconds this block to expire
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to the one set with `setDynBlocksAction()`)

- **warningRate** (*int*) – If set to a non-zero value, the rate above which a warning message will be issued and a no-op block inserted

:**setRCodeRatio**(*rcode*, *ratio*, *seconds*, *reason*, *blockingTime*, *minimumNumberOfResponses*[, *action*[, *warningRate*]])

New in version 1.5.0.

Note: Cache hits are inserted into the in-memory ring buffers since 1.8.0, so they are now considered when computing the rcode ratio.

Adds a rate-limiting rule for the ratio of responses of code `rcode` over the total number of responses for a given client.

Parameters

- **rcode** (*int*) – The response code
- **ratio** (*float*) – Ratio of responses per second of the given rcode over the total number of responses for this client to exceed
- **seconds** (*int*) – Number of seconds the ratio has been exceeded
- **reason** (*string*) – The message to show next to the blocks
- **blockingTime** (*int*) – The number of seconds this block to expire
- **minimumNumberOfResponses** (*int*) – How many total responses is required for this rule to apply
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to the one set with `setDynBlocksAction()`)
- **warningRatio** (*float*) – If set to a non-zero value, the ratio above which a warning message will be issued and a no-op block inserted

:**setQTypeRate**(*qtype*, *rate*, *seconds*, *reason*, *blockingTime*[, *action*[, *warningRate*]])

Adds a rate-limiting rule for queries of type `qtype`, equivalent to: ` `addDynBlocks(exceedQTypeRate(type, rate, seconds), reason, blockingTime, action)` `

Parameters

- **qtype** (*int*) – The qtype
- **rate** (*int*) – Number of queries per second to exceed
- **seconds** (*int*) – Number of seconds the rate has been exceeded
- **reason** (*string*) – The message to show next to the blocks
- **blockingTime** (*int*) – The number of seconds this block to expire
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to the one set with `setDynBlocksAction()`)
- **warningRate** (*int*) – If set to a non-zero value, the rate above which a warning message will be issued and a no-op block inserted

:**setResponseByteRate**(*rate*, *seconds*, *reason*, *blockingTime*[, *action*[, *warningRate*]])

Note: Cache hits are inserted into the in-memory ring buffers since 1.8.0, so they are now considered when computing the bandwidth rate.

Adds a bandwidth rate-limiting rule for responses, equivalent to: ` addDynBlocks(exceedRespByterate(rate, seconds), reason, blockingTime, action) `

Parameters

- **rate** (*int*) – Number of bytes per second to exceed
- **seconds** (*int*) – Number of seconds the rate has been exceeded
- **reason** (*string*) – The message to show next to the blocks
- **blockingTime** (*int*) – The number of seconds this block to expire
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to the one set with `setDynBlocksAction()`)
- **warningRate** (*int*) – If set to a non-zero value, the rate above which a warning message will be issued and a no-op block inserted

:setSuffixMatchRule(*seconds*, *reason*, *blockingTime*, *action*, *visitor*)

New in version 1.4.0.

Changed in version 1.7.0: This visitor function can now optionally return an additional string which will be set as the `reason` for the dynamic block.

Changed in version 1.9.0: This visitor function can now optionally return an additional integer which will be set as the `action` for the dynamic block.

Set a Lua visitor function that will be called for each label of every domain seen in queries and responses. The function receives a `StatNode` object representing the stats of the parent, a `StatNodeStats` one with the stats of the current label and a second `StatNodeStats` with the stats of the current node plus all its children. Note that this function will not be called if a FFI version has been set using `DynBlockRulesGroup:setSuffixMatchRuleFFI()` If the function returns `true`, the current suffix will be added to the block list, meaning that the exact name and all its sub-domains will be blocked according to the *seconds*, *reason*, *blockingTime* and *action* parameters. Since 1.7.0, the function can return an additional string, in addition to the boolean, which will be set as the `reason` for the dynamic block. Selected domains can be excluded from this processing using the `DynBlockRulesGroup:excludeDomains()` method.

This replaces the existing `addDynBlockSMT()` function.

Parameters

- **seconds** (*int*) – Number of seconds the rate has been exceeded
- **reason** (*string*) – The message to show next to the blocks
- **blockingTime** (*int*) – The number of seconds this block to expire
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to the one set with `setDynBlocksAction()`)
- **visitor** (*function*) – The Lua function to call.

:setSuffixMatchRuleFFI(*seconds*, *reason*, *blockingTime*, *action*, *visitor*)

New in version 1.4.0.

Set a Lua FFI visitor function that will be called for each label of every domain seen in queries and responses. The function receives a *dnsdist_ffi_stat_node_t* object containing the stats of the parent, a second one with the stats of the current label and one with the stats of the current node plus all its children. If the function returns `true`, the current suffix will be added to the block list, meaning that the exact name and all its sub-domains will be blocked according to the *seconds*, *reason*, *blockingTime* and *action* parameters. Selected domains can be excluded from this processing using the `DynBlockRulesGroup:excludeDomains()` method.

Parameters

- **seconds** (*int*) – Number of seconds the rate has been exceeded

- **reason** (*string*) – The message to show next to the blocks
- **blockingTime** (*int*) – The number of seconds this block to expire
- **action** (*int*) – The action to take when the dynamic block matches, see *DNSAction*. (default to the one set with `setDynBlocksAction()`)
- **visitor** (*function*) – The Lua FFI function to call.

:apply()

Walk the in-memory query and response ring buffers and apply the configured rate-limiting rules, adding dynamic blocks when the limits have been exceeded.

:setQuiet(*quiet*)

New in version 1.4.0.

Set whether newly blocked clients or domains should be logged.

Parameters **quiet** (*bool*) – True means that insertions will not be logged, false that they will. Default is false.

:excludeDomains(*domains*)

New in version 1.4.0.

Exclude this domain, or list of domains, meaning that no dynamic block will ever be inserted for this domain via `DynBlockRulesGroup:setSuffixMatchRule()` or `DynBlockRulesGroup:setSuffixMatchRuleFFI()`. Default to empty, meaning rules are applied to all domains.

Parameters **domain** (*str*) – A domain, or list of domains, as strings, like for example "powerdns.com"

:excludeRange(*netmasks*)

Changed in version 1.6.0: This method now accepts a `NetmaskGroup` object.

Exclude this range, or list of ranges, meaning that no dynamic block will ever be inserted for clients in that range. Default to empty, meaning rules are applied to all ranges. When used in combination with `DynBlockRulesGroup:includeRange()`, the more specific entry wins.

Parameters **netmasks** (*list*) – A `NetmaskGroup` object, or a netmask or list of netmasks as strings, like for example "192.0.2.1/24"

:includeRange(*netmasks*)

Changed in version 1.6.0: This method now accepts a `NetmaskGroup` object.

Include this range, or list of ranges, meaning that rules will be applied to this range. When used in combination with `DynBlockRulesGroup:excludeRange()`, the more specific entry wins.

Parameters **netmasks** (*list*) – A `NetmaskGroup` object, or a netmask or list of netmasks as strings, like for example "192.0.2.1/24"

:removeRange(*netmasks*)

New in version 1.8.3.

Remove a previously included or excluded range. The range should be an exact match of the existing entry to remove.

Parameters **netmasks** (*list*) – A `NetmaskGroup` object, or a netmask or list of netmasks as strings, like for example "192.0.2.1/24"

:toString()

Return a string describing the rules and range exclusions of this DynBlockRulesGroup.

StatNode

class StatNode

Represent a given node, for the visitor functions used with `DynBlockRulesGroup:setSuffixMatchRule()`

and *DynBlockRulesGroup:setSuffixMatchRuleFFI()*.

fullname
The complete name of that node, ie 'www.powerdns.com.'.

labelsCount
The number of labels in that node, for example 3 for 'www.powerdns.com.'.

:numChildren()
The number of children of that node.

class StatNodeStats
Represent the metrics for a given node, for the visitor functions used with *DynBlockRulesGroup:setSuffixMatchRule()* and *DynBlockRulesGroup:setSuffixMatchRuleFFI()*.

bytes
The number of bytes for all responses returned for that node.

drops
The number of drops for that node.

noerrors
The number of No Error answers returned for that node.

hits
New in version 1.8.0.

The number of cache hits for that node.

nxdomains
The number of NXDomain answers returned for that node.

queries
The number of queries for that node.

servfails
The number of Server Failure answers returned for that node.

SuffixMatchNode

A SuffixMatchNode can be used to quickly check whether a given name belongs to a set or not. This is achieved using an efficient tree structure based on DNS labels, making lookups cheap. Be careful that Suffix Node matching will match for any sub-domain, regardless of the depth, under the name added to the set. For example, if 'example.com.' is added to the set, 'www.example.com.' and 'sub.www.example.com.' will match as well. If you are looking for exact name matching, your might want to consider using a *DNSNameSet* instead.

newSuffixMatchNode()
Creates a new *SuffixMatchNode*.

class SuffixMatchNode
Represent a set of DNS suffixes for quick matching.

:add(*name*)
Changed in version 1.4.0: This method now accepts strings, lists of DNSNames and lists of strings.

Add a suffix to the current set.

Parameters

- **name** (*table*) – The suffix to add to the set.
- **name** – The suffix to add to the set.
- **name** – The suffixes to add to the set. Elements of the table should be of the same type, either DNSName or string.

:check(*name*) → bool

Return true if the given name is a sub-domain of one of those in the set, and false otherwise.

Parameters **name** (DNSName) – The name to test against the set.

:getBestMatch(*name*) → DNSName

New in version 1.8.0.

Returns the best match for the supplied name, or nil if there was no match.

Parameters **name** (DNSName) – The name to look up.

:remove(*name*)

New in version 1.5.0.

Remove a suffix from the current set.

Parameters

- **name** (`table`) – The suffix to remove from the set.
- **name** – The suffix to remove from the set.
- **name** – The suffixes to remove from the set. Elements of the table should be of the same type, either DNSName or string.

19.2.8 Outgoing TLS tickets cache management

Since 1.7, dnsdist supports securing the connection toward backends using DNS over TLS. For these connections, it keeps a cache of TLS tickets to be able to resume a TLS session quickly. By default that cache contains up to 20 TLS tickets per-backend, is cleaned up every 60s, and TLS tickets expire if they have not been used after 600 seconds. These values can be set at configuration time via:

setOutgoingTLSSessionsCacheMaxTicketsPerBackend(*num*)

Set the maximum number of TLS tickets to keep, per-backend, to be able to quickly resume outgoing TLS connections to a backend. Keeping more tickets might provide a better TLS session resumption rate if there is a sudden peak of outgoing connections, at the cost of using a bit more memory.

Parameters **num** (`int`) – The number of TLS tickets to keep, per-backend. The default is 20.

setOutgoingTLSSessionsCacheCleanupDelay(*delay*)

Set the number of seconds between two scans of the TLS sessions cache, removing expired tickets and freeing up memory. Decreasing that value will lead to more scans, freeing up memory more quickly but using a bit more CPU doing so.

Parameters **delay** (`int`) – The number of seconds between two scans of the cache. The default is 60.

setOutgoingTLSSessionsCacheMaxTicketValidity(*validity*)

Set the number of seconds that a given TLS ticket can be kept inactive in the TLS sessions cache. After that delay the ticket will be removed during the next cleanup of the cache. Increasing that value might increase the TLS resumption rate if new connections are not often created, but it might also lead to trying to reuse a ticket that the server will consider too old and refuse.

Parameters **validity** (`int`) – The number of seconds a ticket is considered valid. The default is 600, which matches the default lifetime of TLS tickets set by OpenSSL.

19.2.9 Other functions

addMaintenanceCallback(*callback*)

New in version 1.10.0.

Register a Lua function to be called as part of the `maintenance` hook, which is executed roughly every second. The function should not block for a long period of time, as it would otherwise delay the execution of the other functions registered for this hook, as well as the execution of the *`maintenance()`* function.

Parameters **callback** (*function*) – The function to be called. It takes no parameter and returns no value.

```
function myCallback(hostname, ips)
  print('called')
end
addMaintenanceCallback(myCallback)
```

getAddressInfo(*hostname*, *callback*)

New in version 1.9.0.

Asynchronously resolve, via the system resolver (using `getaddrinfo()`), the supplied `hostname` to IPv4 and IPv6 addresses (if configured on the host) before invoking the supplied `callback` function with the `hostname` and a list of IPv4 and IPv6 addresses as *ComboAddress*. For example, to get the addresses of Quad9's resolver and dynamically add them as backends:

```
function resolveCB(hostname, ips)
  for _, ip in ipairs(ips) do
    newServer(ip:toString())
  end
end
getAddressInfo('dns.quad9.net.', resolveCB)
```

Parameters

- **hostname** (*str*) – The hostname to resolve.
- **callback** (*function*) – The function to invoke when the name has been resolved.

getCurrentTime -> timespec

New in version 1.8.0.

Return the current time, in whole seconds and nanoseconds since epoch.

Returns A timespec object, see *timespec*

getResolvers(*path*)

New in version 1.8.0.

This function can be used to get a Lua table of name servers from a file in the resolv.conf format.

Parameters **path** (*str*) – The path to the file, usually /etc/resolv.conf

getStatisticsCounters()

This function returns a Lua associative array of metrics, with the metric name as key and the current value of the counter as value.

maintenance()

If this function exists, it is called every second to do regular tasks. This can be used for e.g. *Dynamic Blocks*. See also *addMaintenanceCallback()*.

threadmessage(*cmd*, *dict*)

New in version 1.8.0.

This function, if it exists, is called when a separate thread (made with *newThread()*) calls *submitToMainThread()*.

newThread(*code*)

New in version 1.8.0.

Spawns a separate thread running the supplied code. Code is supplied as a string, not as a function object. Note that this function does nothing in 'client' or 'config-check' modes.

submitToMainThread(*cmd*, *dict*)

New in version 1.8.0.

Must be called from a separate thread (made with *newThread()*), submits data to the main thread by calling *threadmessage()* in it. If no `threadmessage` receiver is present in the main thread, `submitToMainThread` logs an error but returns normally.

The `cmd` argument is a string. The `dict` argument is a Lua table.

setAllowEmptyResponse()

New in version 1.4.0.

Set to true (defaults to false) to allow empty responses (qdcount=0) with a NoError or NXDomain rcode (default) from backends. dnsdist drops these responses by default because it can't match them against the initial query since they don't contain the qname, qtype and qclass, and therefore the risk of collision is much higher than with regular responses.

setDropEmptyQueries(*drop*)

New in version 1.6.0.

Set to true (defaults to false) to drop empty queries (qdcount=0) right away, instead of answering with a NotImp rcode. dnsdist used to drop these queries by default because most rules and existing Lua code expects a query to have a qname, qtype and qclass. However **RFC 7873** uses these queries to request a server cookie, and **RFC 8906** as a conformance test, so answering these queries with NotImp is much better than not answering at all.

Parameters **drop** (*bool*) – Whether to drop these queries (defaults to false)

setProxyProtocolMaximumPayloadSize(*size*)

New in version 1.6.0.

Set the maximum size of a Proxy Protocol payload that dnsdist is willing to accept, in bytes. The default is 512, which is more than enough except for very large TLV data. This setting can't be set to a value lower than 16 since it would deny of Proxy Protocol headers.

Parameters **size** (*int*) – The maximum size in bytes (default is 512)

setTCPFastOpenKey(*key*)

New in version 1.8.0.

Set the supplied `TCP Fast Open` key on all frontends. This can for example be used to allow all dnsdist instances in an anycast cluster to use the same `TCP Fast Open` key, reducing round-trips.

Parameters **key** (*string*) – The format of the key can be found in `/proc/sys/net/ipv4/tcp_fastopen_key`

makeIPCipherKey(*password*) → string

New in version 1.4.0.

Hashes the password to generate a 16-byte key that can be used to pseudonymize IP addresses with IP cipher.

generateOCSPResponse(*pathToServerCertificate*, *pathToCACertificate*, *pathToCAPrivateKey*, *outputFile*, *numberOfDaysOfValidity*, *numberOfMinutesOfValidity*)

New in version 1.4.0.

When a local PKI is used to issue the certificate, or for testing purposes, *generateOCSPResponse()* can be used to generate an OCSP response file for a certificate, using the certificate and private key of the certification authority that signed that certificate. The resulting file can be directly used with the *addDOHLocal()* or the *addTLSLocal()* functions.

Parameters

- **pathToServerCertificate** (*string*) – Path to a file containing the certificate used by the server.
- **pathToCACertificate** (*string*) – Path to a file containing the certificate of the certification authority that was used to sign the server certificate.
- **pathToCAPrivateKey** (*string*) – Path to a file containing the private key corresponding to the certification authority certificate.

- **outputFile** (*string*) – Path to a file where the resulting OCSP response will be written to.
- **numberOfDaysOfValidity** (*int*) – Number of days this OCSP response should be valid.
- **numberOfMinutesOfValidity** (*int*) – Number of minutes this OCSP response should be valid, in addition to the number of days.

getRingEntries()

New in version 1.8.0.

Return a list of all the entries, queries and responses alike, that are present in the in-memory ring buffers, as *LuaRingEntry* objects.

loadTLSEngine(*engineName*[, *defaultString*])

New in version 1.8.0.

Load the OpenSSL engine named `engineName`, setting the engine default string to `defaultString` if supplied. Engines can be used to accelerate cryptographic operations, like for example Intel QAT. At the moment up to a maximum of 32 loaded engines are supported, and that support is experimental. Some engines might actually degrade performance unless the TLS asynchronous mode of OpenSSL is enabled. To enable it see the `tlsAsyncMode` parameter on *addTLSLocal()* and *addDOHLocal()*.

Parameters

- **engineName** (*string*) – The name of the engine to load.
- **defaultString** (*string*) – The default string to pass to the engine. The exact value depends on the engine but represents the algorithms to register with the engine, as a list of comma-separated keywords. For example "RSA,EC,DSA,DH,PKEY,PKEY_CRYPTO,PKEY_ASN1".

loadTLSProvider(*providerName*)

New in version 1.8.0.

Load the OpenSSL provider named `providerName`. Providers can be used to accelerate cryptographic operations, like for example Intel QAT. At the moment up to a maximum of 32 loaded providers are supported, and that support is experimental. Note that *loadTLSProvider()* is only available when building against OpenSSL version >= 3.0 and with the *–enable-tls-provider* configure flag on. In other cases, *loadTLSEngine()* should be used instead. Some providers might actually degrade performance unless the TLS asynchronous mode of OpenSSL is enabled. To enable it see the `tlsAsyncMode` parameter on *addTLSLocal()* and *addDOHLocal()*.

Parameters **providerName** (*string*) – The name of the provider to load.

newTLSCertificate(*pathToCert*[, *options*])

New in version 1.8.0.

Creates a *TLSCertificate* object suited to be used with functions like *addDOHLocal()*, *addDOH3Local()*, *addDOQLocal()* and *addTLSLocal()* for TLS certificate configuration.

`PKCS12` files are only supported by the `openssl` provider, password-protected or not.

Parameters

- **pathToCert** (*string*) – Path to a file containing the certificate or a `PKCS12` file containing both a certificate and a key.
- **options** (*table*) – A table with key: value pairs with additional options.

Options:

- `key="path/to/key"`: string - Path to a file containing the key corresponding to the certificate.
- `password="pass"`: string - Password protecting the `PKCS12` file if appropriate.

```
newTLSCertificate("path/to/pub.crt", {key="path/to/private.pem"})
newTLSCertificate("path/to/domain.p12", {password="passphrase"}) -- use a␣
↪password protected ``PKCS12`` file
```

DOHFrontend

class DOHFrontend

New in version 1.4.0.

This object represents an address and port dnsdist is listening on for DNS over HTTPS queries.

:getAddressAndPort() → string

New in version 1.7.1.

Return the address and port this frontend is listening on.

:loadNewCertificatesAndKeys(*certFile(s)*, *keyFile(s)*)

New in version 1.6.1.

Changed in version 1.8.0: `certFile` now accepts a TLSCertificate object or a list of such objects (see *newTLSCertificate()*)

Parameters

- **certFile(s)** (*str*) – The path to a X.509 certificate file in PEM format, a list of paths to such files, or a TLSCertificate object.
- **keyFile(s)** (*str*) – The path to the private key file corresponding to the certificate, or a list of paths to such files, whose order should match the certFile(s) ones. Ignored if `certFile` contains TLSCertificate objects.

:loadTicketsKeys(*ticketsKeysFile*)

Load new tickets keys from the selected file, replacing the existing ones. These keys should be rotated often and never written to persistent storage to preserve forward secrecy. The default is to generate a random key. dnsdist supports several tickets keys to be able to decrypt existing sessions after the rotation. See *TLS Sessions Management* for more information.

Parameters **ticketsKeysFile** (*str*) – The path to a file from where TLS tickets keys should be loaded.

:reloadCertificates()

Reload the current TLS certificate and key pairs.

:rotateTicketsKey()

Replace the current TLS tickets key by a new random one.

:setResponsesMap(*rules*)

Set a list of HTTP response rules allowing to intercept HTTP queries very early, before the DNS payload has been processed, and send custom responses including error pages, redirects and static content.

Parameters of DOHResponseMapEntry objects rules (*list*) – A list of DOHResponseMapEntry objects, obtained with *newDOHResponseMapEntry()*.

newDOHResponseMapEntry(*regex*, *status*, *content*[, *headers*]) → DOHResponseMapEntry

New in version 1.4.0.

Return a DOHResponseMapEntry that can be used with *DOHFrontend:setResponsesMap()*. Every query whose path is listed in the `urls` parameter to *addDOHLocal()* and matches the regular expression supplied in `regex` will be immediately answered with a HTTP response. The status of the HTTP response will be the one supplied by `status`, and the content set to the one supplied by `content`, except if the status is a redirection (3xx) in which case the content is expected to be the URL to redirect to.

Parameters

- **regex** (*str*) – A regular expression to match the path against.
- **status** (*int*) – The HTTP code to answer with.
- **content** (*str*) – The content of the HTTP response, or a URL if the status is a redirection (3xx).
- **of headers** (*table*) – The custom headers to set for the HTTP response, if any. The default is to use the value of the `customResponseHeaders` parameter passed to `addDOHLocal()`.

DOH3Frontend

class DOH3Frontend

New in version 1.9.0.

This object represents an address and port dnsdist is listening on for DNS over HTTP3 queries.

:reloadCertificates()

Reload the current TLS certificate and key pairs.

DOQFrontend

class DOQFrontend

New in version 1.9.0.

This object represents an address and port dnsdist is listening on for DNS over QUIC queries.

:reloadCertificates()

Reload the current TLS certificate and key pairs.

LuaRingEntry

class LuaRingEntry

New in version 1.8.0.

This object represents an entry from the in-memory ring buffers, query or response.

backend

If this entry is a response, the backend from which it has been received as a *ComboAddress*.

`LuaRingEntry.`**dnsheader**

The *DNSHeader (dh) object* of this entry.

isResponse

Whether this entry is a response (true) or a request (false).

macAddress

The MAC address of the client as a string, if available.

protocol

The protocol (Do53 UDP, Do53 TCP, DoT, DoH, . . .) over which this entry was received, as a string.

qname

The qname of this entry as a *DNSName objects*.

qtype

The qtype of this entry as an integer.

requestor

The requestor (client IP) of this entry as a *ComboAddress*.

size

The size of the DNS payload of that entry, in bytes.

`LuaRingEntry.`**`usec`**

The response time (elapsed time between the request was received and the response sent) in milliseconds.

`LuaRingEntry.`**`when`**

The timestamp of this entry, as a *timespec*.

timespec

`class timespec`

New in version 1.8.0.

This object represents a timestamp in the timespec format.

`tv_sec`

Number of seconds elapsed since Unix epoch.

`tv_nsec`

Number of remaining nanoseconds elapsed since Unix epoch after subtracting the seconds from the *tv_sec* field.

TLSCertificate

`class TLSCertificate`

This object represents a TLS certificate. It can be created with *`newTLSCertificate()`* and used with *`addDOHLocal()`*, *`addDOH3Local()`*, *`addDOQLocal()`* and *`addTLSLocal()`* for TLS certificate configuration. It is mostly useful to deal with password-protected `PKCS12` certificates.

TLSContext

`class TLSContext`

This object represents an address and port dnsdist is listening on for DNS over TLS queries.

`:loadTicketsKeys``(`*ticketsKeysFile*`)`

Load new tickets keys from the selected file, replacing the existing ones. These keys should be rotated often and never written to persistent storage to preserve forward secrecy. The default is to generate a random key. The OpenSSL provider supports several tickets keys to be able to decrypt existing sessions after the rotation, while the GnuTLS provider only supports one key. See *TLS Sessions Management* for more information.

Parameters **`ticketsKeysFile`** (*`str`*) – The path to a file from where TLS tickets keys should be loaded.

`:rotateTicketsKey``()`

Replace the current TLS tickets key by a new random one.

TLSFrontend

`class TLSFrontend`

This object represents the configuration of a listening frontend for DNS over TLS queries. To each frontend is associated a TLSContext.

`:getAddressAndPort``()` → string

New in version 1.7.1.

Return the address and port this frontend is listening on.

:loadNewCertificatesAndKeys(*certFile(s)*, *keyFile(s)*)

Create and switch to a new TLS context using the same options than were passed to the corresponding *addTLSLocal()* directive, but loading new certificates and keys from the selected files, replacing the existing ones.

Parameters

- **certFile(s)** (*str*) – The path to a X.509 certificate file in PEM format, or a list of paths to such files.
- **keyFile(s)** (*str*) – The path to the private key file corresponding to the certificate, or a list of paths to such files, whose order should match the certFile(s) ones.

:loadTicketsKeys(*ticketsKeysFile*)

New in version 1.6.0: Load new tickets keys from the selected file, replacing the existing ones. These keys should be rotated often and never written to persistent storage to preserve forward secrecy. The default is to generate a random key. The OpenSSL provider supports several tickets keys to be able to decrypt existing sessions after the rotation, while the GnuTLS provider only supports one key. See *TLS Sessions Management* for more information.

param str ticketsKeysFile The path to a file from where TLS tickets keys should be loaded.

:reloadCertificates()

New in version 1.6.0: Reload the current TLS certificate and key pairs.

:rotateTicketsKey()

New in version 1.6.0: Replace the current TLS tickets key by a new random one.

EDNS on Self-generated answers

There are several mechanisms in dnsdist that turn an existing query into an answer right away, without reaching out to the backend, including *SpoofAction()*, *RCodeAction()*, *TCAction()* and returning a response from Lua. Those responses should, according to **RFC 6891**, contain an OPT record if the received request had one, which is the case by default and can be disabled using *setAddEDNSToSelfGeneratedResponses()*.

We must, however, provide a responder's maximum payload size in this record, and we can't easily know the maximum payload size of the actual backend so we need to provide one. The default value is 1232 since 1.6.0, and can be overridden using *setPayloadSizeOnSelfGeneratedAnswers()*.

setAddEDNSToSelfGeneratedResponses(*add*)

Whether to add EDNS to self-generated responses, provided that the initial query had EDNS.

Parameters **add** (*bool*) – Whether to add EDNS, default is true.

setPayloadSizeOnSelfGeneratedAnswers(*payloadSize*)

Changed in version 1.6.0: Default value changed from 1500 to 1232.

Set the UDP payload size advertised via EDNS on self-generated responses. In accordance with **RFC 6891**, values lower than 512 will be treated as equal to 512.

Parameters **payloadSize** (*int*) – The responder's maximum UDP payload size, in bytes. Default is 1232 since 1.6.0, it was 1500 before.

Security Polling

PowerDNS products can poll the security status of their respective versions. This polling, naturally, happens over DNS. If the result is that a given version has a security problem, the software will report this at level 'Error' during startup, and repeatedly during operations, every *setSecurityPollInterval()* seconds.

By default, security polling happens on the domain 'secpoll.powerdns.com', but this can be changed with the `setSecurityPollSuffix()` function. If this setting is made empty, no polling will take place. Organizations wanting to host their own security zones can do so by changing this setting to a domain name under their control.

To enable distributors of PowerDNS to signal that they have backported versions, the PACKAGEVERSION compilation-time macro can be used to set a distributor suffix.

setSecurityPollInterval(*interval*)

Set the interval, in seconds, between two security polls.

Parameters **interval** (*int*) – The interval, in seconds, between two polls. Default is 3600.

setSecurityPollSuffix(*suffix*)

Domain name from which to query security update notifications. Setting this to an empty string disables secpoll.

Parameters **suffix** (*string*) – The suffix to use, default is 'secpoll.powerdns.com.'.

19.3 Constants

There are many constants in **dnsdist**.

19.3.1 OPCode

These constants represent the OpCode of a query.

- `DNSOpcode.Query`
- `DNSOpcode.IQuery`
- `DNSOpcode.Status`
- `DNSOpcode.Notify`
- `DNSOpcode.Update`

Reference: https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-5

19.3.2 DNSClass

These constants represent the CLASS of a DNS record.

- `DNSClass.IN`
- `DNSClass.CHAOS`
- `DNSClass.NONE`
- `DNSClass.ANY`

Reference: https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-2

19.3.3 RCode

These constants represent the different RCODEs for DNS messages.

Changed in version 1.4.0: The prefix is changed from `dnsdist` to `DNSRCode`.

Changed in version 1.7.0: The lookup fallback from `dnsdist` to `DNSRCode` was removed.

- `DNSRCode.NOERROR`
- `DNSRCode.FORMERR`

- DNSRCode.SERVFAIL
- DNSRCode.NXDOMAIN
- DNSRCode.NOTIMP
- DNSRCode.REFUSED
- DNSRCode.YXDOMAIN
- DNSRCode.YXRRSET
- DNSRCode.NXRRSET
- DNSRCode.NOTAUTH
- DNSRCode.NOTZONE

RCodes below are extended RCodes that can only be matched using `ERCodeRule()`.

- DNSRCode.BADVERS
- DNSRCode.BADSIG
- DNSRCode.BADKEY
- DNSRCode.BADTIME
- DNSRCode.BADMODE
- DNSRCode.BADNAME
- DNSRCode.BADALG
- DNSRCode.BADTRUNC
- DNSRCode.BADCOOKIE

19.3.4 EDNSOptionCode

- EDNSOptionCode.DHU
- EDNSOptionCode.ECS
- EDNSOptionCode.N3U
- EDNSOptionCode.DAU
- EDNSOptionCode.TCPKEEPALIVE
- EDNSOptionCode.COOKIE
- EDNSOptionCode.PADDING
- EDNSOptionCode.KEYTAG
- EDNSOptionCode.NSID
- EDNSOptionCode.CHAIN
- EDNSOptionCode.EXPIRE

Reference: https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-11

19.3.5 DNS Packet Sections

These constants represent the section in the DNS Packet.

- DNSSection.Question
- DNSSection.Answer

- DNSSection.Authority
- DNSSection.Additional

19.3.6 DNSAction

Changed in version 1.5.0: DNSAction.SpoofRaw has been added.

Changed in version 1.8.0: DNSAction.SpoofPacket has been added.

These constants represent an Action that can be returned from *LuaAction()* functions.

- DNSAction.Allow: let the query pass, skipping other rules
- DNSAction.Delay: delay the response for the specified milliseconds (UDP-only), continue to the next rule
- DNSAction.Drop: drop the query
- DNSAction.HeaderModify: indicate that the query has been turned into a response
- DNSAction.None: continue to the next rule
- DNSAction.NoOp: continue to the next rule (used for Dynamic Block actions where None has a different meaning)
- DNSAction.Nxdomain: return a response with a NXDomain rcode
- DNSAction.Pool: use the specified pool to forward this query
- DNSAction.Refused: return a response with a Refused rcode
- DNSAction.ServFail: return a response with a ServFail rcode
- DNSAction.Spoof: spoof the response using the supplied IPv4 (A), IPv6 (AAAA) or string (CNAME) value. TTL will be 60 seconds.
- DNSAction.SpoofPacket: spoof the response using the supplied raw packet
- DNSAction.SpoofRaw: spoof the response using the supplied raw value as record data (see also *DNSQuestion:spoof()* and dnsdist_ffi_dnsquestion_spoof_raw() to spoof multiple values)
- DNSAction.Truncate: truncate the response
- DNSAction.NoRecurse: set rd=0 on the query

19.3.7 DNSQType

Changed in version 1.4.0: The prefix is changed from dnsdist. to DNSQType.

Changed in version 1.7.0: The lookup fallback from dnsdist to DNSQType was removed.

All named QTypes are available as constants, prefixed with DNSQType., e.g.:

- DNSQType.AAAA
- DNSQType.AXFR
- DNSQType.A
- DNSQType.NS
- DNSQType.SOA
- etc.

19.3.8 DNSResponseAction

Changed in version 1.9.0: The `DNSResponseAction.Truncate` value was added.

These constants represent an Action that can be returned from *`LuaResponseAction()`* functions.

- `DNSResponseAction.Allow`: let the response pass, skipping other rules
- `DNSResponseAction.Delay`: delay the response for the specified milliseconds (UDP-only), continue to the next rule
- `DNSResponseAction.Drop`: drop the response
- `DNSResponseAction.HeaderModify`: indicate that the query has been turned into a response
- `DNSResponseAction.None`: continue to the next rule
- `DNSResponseAction.ServFail`: return a response with a ServFail rcode
- `DNSResponseAction.Truncate`: truncate the response, removing all records from the answer, authority and additional sections if any

19.4 ComboAddress

IP addresses are moved around in a native format, called a *`ComboAddress`*. ComboAddresses can be IPv4 or IPv6, and unless you want to know, you don't need to.

`newCA`(*address*) → ComboAddress

Returns a *`ComboAddress`* based on `address`

Parameters **`address`** (*`string`*) – The IP address, with optional port, to represent.

`class ComboAddress`

A `ComboAddress` represents an IP address with possibly a port number. The object can be an IPv4 or an IPv6 address. It has these methods:

`:getPort`() → int

Returns the port number.

`:ipdecrypt`(*key*) → ComboAddress

Decrypt this IP address as described in https://powerdns.org/ipcipher

Parameters **`key`** (*`string`*) – A 16 byte key. Note that this can be derived from a passphrase with the standalone function *makeIPCipherKey*

`:ipencrypt`(*key*) → ComboAddress

Encrypt this IP address as described in https://powerdns.org/ipcipher

Parameters **`key`** (*`string`*) – A 16 byte key. Note that this can be derived from a passphrase with the standalone function *makeIPCipherKey*

`:isIPv4`() → bool

Returns true if the address is an IPv4, false otherwise

`:isIPv6`() → bool

Returns true if the address is an IPv6, false otherwise

`:isMappedIPv4`() → bool

Returns true if the address is an IPv4 mapped into an IPv6, false otherwise

`:mapToIPv4`() → ComboAddress

Convert an IPv4 address mapped in a v6 one into an IPv4. Returns a new *`ComboAddress`*

`:tostring`() → string
`:toString`() → string

Returns in human-friendly format

:tostringWithPort() → string
:toStringWithPort() → string

Returns in human-friendly format, with port number

:truncate(*bits*)

Truncate the *ComboAddress* to the specified number of bits. This essentially zeroes all bits after `bits`.

Parameters **bits** (*int*) – Amount of bits to truncate to

19.5 Netmask

newNetmask(*str*) → Netmask
newNetmask(*ca*, *bits*) → Netmask

New in version 1.5.0.

Returns a Netmask

Parameters

- **str** (*string*) – A netmask, like `192.0.2.0/24`.
- **ca** (ComboAddress) – A *ComboAddress*.
- **bits** (*int*) – The number of bits in this netmask.

class Netmask

New in version 1.5.0: Represents a netmask.

:getBits() → int

Return the number of bits of this netmask, for example `24` for `192.0.2.0/24`.

:getMaskedNetwork() → ComboAddress

Return a *ComboAddress* object representing the base network of this netmask object after masking any additional bits if necessary (for example `192.0.2.0` if the netmask was constructed with `newNetmask('192.0.2.1/24')`).

:empty() → bool

Return true if the netmask is empty, meaning that the netmask has not been set to a proper value.

:isIPv4() → bool

Return true if the netmask is an IPv4 one.

:isIPv6() → bool

Return true if the netmask is an IPv6 one.

:getNetwork() → ComboAddress

Return a *ComboAddress* object representing the base network of this netmask object.

:match(*str*) → bool

Return true if the address passed in the `str` parameter belongs to this netmask.

Parameters **str** (*string*) – A network address, like `192.0.2.0`.

:toString() → string

Return a string representation of the netmask, for example `192.0.2.0/24`.

19.6 NetmaskGroup

newNMG() → NetmaskGroup

Returns a NetmaskGroup

class NetmaskGroup

Represents a group of netmasks that can be used to match *ComboAddress*es against.

:addMask(*mask*)
:addMasks(*masks*)
Add one or more masks to the NMG.

Parameters

- **mask** (*string*) – Add this mask, prefix with *!* to exclude this mask from matching.
- **masks** (*table*) – Adds the keys of the table to the *NetmaskGroup*. It should be a table whose keys are *ComboAddress* objects and whose values are integers. The integer values of the table entries are ignored. The table is of the same type as the table returned by the *exceed** functions.

:addNMG(*otherNMG*)
New in version 1.9.0.

Add all masks from an existing NMG to this NMG.

Parameters **otherNMG** (NetmaskGroup) – Add the masks from a *NetmaskGroup* to this one.

:match(*address*) → bool
Checks if `address` is matched by this NetmaskGroup.

Parameters **address** (ComboAddress) – The address to match.

:clear()
Clears the NetmaskGroup.

:size() → int
Returns number of netmasks in this NetmaskGroup.

19.7 DNSName objects

A *DNSName* object represents a name in the DNS. It has several functions that can manipulate it without conversions to strings. Creating a `DNSName` is done with the *newDNSName()*:

```
myname = newDNSName("www.example.com")
```

dnsdist will complain loudly if the name is invalid (e.g. too long, dot in the wrong place).

The `myname` variable has several functions to get information from it

```
print(myname:countLabels()) -- prints "3"
print(myname:wirelength()) -- prints "17"
name2 = newDNSName("example.com")
if myname:isPartOf(name2) then -- prints "it is"
  print('it is')
end
```

19.7.1 Functions and methods of a DNSName

newDNSName(*name*) → DNSName
Returns the *DNSName* object of `name`.

Parameters **name** (*string*) – The name to create a DNSName for

class DNSName
A `DNSName` object represents a name in the DNS. It is returned by several functions and has several functions to programmatically interact with it.

:chopOff() → bool
Removes the left-most label and returns `true`. `false` is returned if no label was removed

:countLabels() → int
Returns the number of DNSLabels in the name

:isPartOf(*name*) → bool
Returns true if the DNSName is part of the DNS tree of `name`.

Parameters **name** (`DNSName`) – The name to check against

:makeRelative(*name*) → DNSName
New in version 1.8.0.

Provided that the current name is part of the supplied name, returns a new DNSName composed only of the labels that are below the supplied name (ie making www.powerdns.com relative to powerdns.com would return only wwww) Otherwise an empty (unset) DNSName is returned.

Parameters **name** (`DNSName`) – The name to make us relative against

:toDNSString() → string
Returns a wire format form of the DNSName, suitable for usage in *SpoofRawAction()*.

:toString() → string
:tostring() → string
Returns a human-readable form of the DNSName.

:toStringNoDot() → string
New in version 1.8.0.

Returns a human-readable form of the DNSName, without the trailing dot.

:wirelength() → int
Returns the length in bytes of the DNSName as it would be on the wire.

19.8 DNSNameSet objects

A *DNSNameSet* object is a set of *DNSName* objects. Based on std::unordered_set (hash table). Creating a `DNSName` is done with the *newDNSNameSet()*:

```
myset = newDNSNameSet()
```

The set can be filled by func:*DNSNameSet:add*:

```
myset:add(newDNSName("domain1.tld"))
myset:add(newDNSName("domain2.tld"))
```

19.8.1 Functions and methods of a DNSNameSet

newDNSNameSet() → DNSNameSet
Returns the *DNSNameSet*.

class DNSNameSet
A `DNSNameSet` object is a set of *DNSName* objects.

:add(*name*)
Adds the name to the set.

Parameters **name** (`DNSName`) – The name to add.

:empty() → bool
Returns true is the DNSNameSet is empty.

:clear()
Clean up the set.

:toString() → string
Returns a human-readable form of the DNSNameSet.

:size() → int
Returns the number of names in the set.

:delete(*name*) → int
Removes the name from the set. Returns the number of deleted elements.

Parameters **name** (DNSName) – The name to remove.

:check(*name*) → bool
Returns true if the set contains the name.

Parameters **name** (DNSName) – The name to check.

19.9 The DNSQuestion (dq) object

A DNSQuestion or `dq` object is available in several hooks and Lua actions. This object contains details about the current state of the question. This state can be modified from the various hooks.

class DNSQuestion
The DNSQuestion object has several attributes, many of them read-only:

deviceID
New in version 1.8.0.

The identifier of the remote device, which will be exported via ProtoBuf if set.

deviceName
New in version 1.8.0.

The name of the remote device, which will be exported via ProtoBuf if set.

dh
The *DNSHeader (dh) object* of this query.

ecsOverride
Whether an existing ECS value should be overridden, settable.

ecsPrefixLength
The ECS prefix length to use, settable.

len
The length of the data starting at *DNSQuestion.dh*, including any trailing bytes following the DNS message.

localaddr
ComboAddress of the local bind this question was received on.

opcode
Integer describing the OPCODE of the packet. Can be matched against *OPCode*.

pool
New in version 1.8.0.

The pool of servers to which this query will be routed.

qclass
QClass (as an unsigned integer) of this question. Can be compared against *DNSClass*.

qname
DNSName of this question.

qtype
QType (as an unsigned integer) of this question. Can be compared against the pre-defined *constants* like `DNSQType.A`, `DNSQType.AAAA`.

remoteaddr
ComboAddress of the remote client.

requestorID
New in version 1.8.0.

The identifier of the requestor, which will be exported via ProtoBuf if set.

rcode
RCode (as an unsigned integer) of this question. Can be compared against *RCode*

size
The total size of the buffer starting at *`DNSQuestion.dh`*.

skipCache
Whether to skip cache lookup / storing the answer for this question, settable.

tempFailureTTL
On a SERVFAIL or REFUSED from the backend, cache for this amount of seconds, settable.

tcp
Whether the query was received over TCP.

useECS
Whether to send ECS to the backend, settable.

It also supports the following methods:

:addProxyProtocolValue(*type*, *value*)
New in version 1.6.0.

Add a proxy protocol TLV entry of type `type` and `value` to the current query.

Parameters

- **type** (*int*) – The type of the new value, ranging from 0 to 255 (both included)
- **value** (*str*) – The binary-safe value

:getContent() → str
New in version 1.8.0.

Get the content of the DNS packet as a string

:getDO() → bool
Get the value of the DNSSEC OK bit.

Returns true if the DO bit was set, false otherwise

:getEDNSOptions() → table
Return the list of EDNS Options, if any.

Returns A table of EDNSOptionView objects, indexed on the ECS Option code

:getHTTPHeaders() → table
New in version 1.4.0.

Changed in version 1.8.0: see `keepIncomingHeaders` on *`addDOHLocal()`*

Return the HTTP headers for a DoH query, as a table whose keys are the header names and values the header values. Since 1.8.0 it is necessary to set the `keepIncomingHeaders` option to true on *`addDOHLocal()`* to be able to use this method.

Returns A table of HTTP headers

:getHTTPHost() → string
New in version 1.4.0.

Return the HTTP Host for a DoH query, which may or may not contain the port.

Returns The host of the DoH query

:getHTTPPath() → string
New in version 1.4.0.

Return the HTTP path for a DoH query.

Returns The path part of the DoH query URI

:getHTTPQueryString() → string
New in version 1.4.0.

Return the HTTP query string for a DoH query.

Returns The query string part of the DoH query URI

:getHTTPScheme() → string
New in version 1.4.0.

Return the HTTP scheme for a DoH query.

Returns The scheme of the DoH query, for example `http` or `https`

:getProtocol() → string
New in version 1.7.0.

Return the transport protocol this query was received over, as a string. The possible values are:

- "Do53 UDP"
- "Do53 TCP"
- "DNSCrypt UDP"
- "DNSCrypt TCP"
- "DNS over TLS"
- "DNS over HTTPS"

Returns A string

:getProxyProtocolValues() → table
New in version 1.6.0.

Return a table of the Proxy Protocol values currently set for this query.

Returns A table whose keys are types and values are binary-safe strings

DNSQuestion:getQueryTime -> timespec
New in version 1.8.0.

Return the time at which the current query has been received, in whole seconds and nanoseconds since epoch, as a *timespec* object.

Returns A *timespec* object

:getServerNameIndication() → string
New in version 1.4.0.

Return the TLS Server Name Indication (SNI) value sent by the client over DoT or DoH, if any. See *SNIRule()* for more information, especially about the availability of SNI over DoH.

Returns A string containing the TLS SNI value, if any

:getTag(*key*) → string
Get the value of a tag stored into the DNSQuestion object.

Parameters **key** (*string*) – The tag's key

Returns The tag's value if it was set, an empty string otherwise

:getTagArray() → table

Get all the tags stored into the DNSQuestion object.

Returns A table of tags, using strings as keys and values

:getTrailingData() → string

New in version 1.4.0.

Get all data following the DNS message.

Returns The trailing data as a null-safe string

:changeName(*newName*) → bool

New in version 1.8.0.

Change the qname of the current query in the DNS payload. The reverse operation will have to be done on the response to set it back to the initial name, or the client will be confused and likely drop the response. See *DNSResponse:changeName()*. Returns false on failure, true on success.

Parameters **newName** (DNSName) – The new qname to use

:sendTrap(*reason*)

Send an SNMP trap.

Parameters **reason** (*string*) – An optional string describing the reason why this trap was sent

:setContent(*data*)

New in version 1.8.0.

Replace the whole DNS payload of the query with the supplied data. The new DNS payload must include the DNS header, whose ID will be adjusted to match the one of the existing query. For example, this replaces the whole DNS payload of queries for custom.async.tests.powerdns.com and type A, turning it them into `FORMERR` responses, including EDNS with the `DNSSECOK` bit set and a UDP payload size of 1232:

```
function replaceQueryPayload(dq)
  local raw =
↪'\000\000\128\129\000\001\000\000\000\000\000\001\006custom\005async\005tests\008powerd
↪'
  dq:setContent(raw)
  return DNSAction.Allow
end
addAction(AndRule({QTypeRule(DNSQType.A), makeRule('custom.async.tests.
↪powerdns.com')}), LuaAction(replaceQueryPayload))
```

Parameters **data** (*string*) – The raw DNS payload

:setEDNSOption(*code*, *data*)

New in version 1.8.0.

Add arbitrary EDNS option and data to the query. Any existing EDNS content with the same option code will be overwritten.

Parameters

- **code** (*int*) – The EDNS option code
- **data** (*string*) – The EDNS option raw data

:setExtendedDNSError(*infoCode*[, *extraText*])

New in version 1.9.0: Set an Extended DNS Error status that will be added to the response corresponding to the current query.

Parameters

- **infoCode** (*int*) – The EDNS Extended DNS Error code
- **extraText** (*string*) – The optional EDNS Extended DNS Error extra text

:**setHTTPResponse**(*status*, *body*, *contentType=""*)

New in version 1.4.0.

Set the HTTP status code and content to immediately send back to the client. For HTTP redirects (3xx), the string supplied in `body` should be the URL to redirect to. For 200 responses, the value of the content type header can be specified via the `contentType` parameter. In order for the response to be sent, the QR bit should be set before returning and the function should return Action.HeaderModify.

Parameters

- **status** (*int*) – The HTTP status code to return
- **body** (*string*) – The body of the HTTP response, or a URL if the status code is a redirect (3xx)
- **contentType** (*string*) – The HTTP Content-Type header to return for a 200 response, ignored otherwise. Default is `application/dns-message`.

:**setNegativeAndAdditionalSOA**(*nxd*, *zone*, *ttl*, *mname*, *rname*, *serial*, *refresh*, *retry*, *expire*, *minimum*)

New in version 1.5.0.

Turn a question into a response, either a NXDOMAIN or a NODATA one based on `nxd`, setting the QR bit to 1 and adding a SOA record in the additional section.

Parameters

- **nxd** (*bool*) – Whether the answer is a NXDOMAIN (true) or a NODATA (false)
- **zone** (*string*) – The owner name for the SOA record
- **ttl** (*int*) – The TTL of the SOA record
- **mname** (*string*) – The mname of the SOA record
- **rname** (*string*) – The rname of the SOA record
- **serial** (*int*) – The value of the serial field in the SOA record
- **refresh** (*int*) – The value of the refresh field in the SOA record
- **retry** (*int*) – The value of the retry field in the SOA record
- **expire** (*int*) – The value of the expire field in the SOA record
- **minimum** (*int*) – The value of the minimum field in the SOA record

:**setProxyProtocolValues**(*values*)

New in version 1.5.0.

Set the Proxy-Protocol Type-Length values to send to the backend along with this query.

Parameters **values** (*table*) – A table of types and values to send, for example: `{ [0x00] = "foo", [0x42] = "bar" }`. Note that the type must be an integer. Try to avoid these values: 0x01 - 0x05, 0x20 - 0x25, 0x30 as those are predefined in https://www.haproxy.org/download/2.3/doc/proxy-protocol.txt (search for *PP2_TYPE_ALPN*)

:**setRestartable**()

New in version 1.8.0.

Make it possible to restart that query after receiving the response, for example to try a different pool of servers after receiving a SERVFAIL or a REFUSED response. Under the hood, this tells dnsdist to keep a copy of the initial query around so that we can send it a second time if needed. Copying the initial DNS payload has a small memory and CPU cost and thus is not done by default. See also `DNSResponse:restart()`.

:setTag (*key*, *value*)

Changed in version 1.7.0: Prior to 1.7.0 calling `DNSQuestion:setTag()` would not overwrite an existing tag value if already set.

Set a tag into the DNSQuestion object. Overwrites the value if any already exists.

Parameters

- **key** (*string*) – The tag's key
- **value** (*string*) – The tag's value

:setTagArray (*tags*)

Changed in version 1.7.0: Prior to 1.7.0 calling `DNSQuestion:setTagArray()` would not overwrite existing tag values if already set.

Set an array of tags into the DNSQuestion object. Overwrites the values if any already exist.

Parameters **tags** (*table*) – A table of tags, using strings as keys and values

:setTrailingData (*tail*) → bool

New in version 1.4.0.

Set the data following the DNS message, overwriting anything already present.

Parameters **tail** (*string*) – The new data

Returns true if the operation succeeded, false otherwise

:spoof (*ip|ips|raw|raws*[, *typeForAny*])

New in version 1.6.0.

Changed in version 1.9.0: Optional parameter `typeForAny` added.

Forge a response with the specified record data as raw bytes. If you specify list of raws (it is assumed they match the query type), all will get spoofed in.

Parameters

- **ip** (ComboAddress) – The *ComboAddress* to be spoofed, e.g. *newCA("192.0.2.1")*.
- **ComboAddresses ips** (*table*) – The *ComboAddress'es to be spoofed, e.g. '{ newCA("192.0.2.1"), newCA("192.0.2.2") }*.
- **raw** (*string*) – The raw string to be spoofed, e.g. *"\192\000\002\001"*.
- **raws** (*table*) – The raw strings to be spoofed, e.g. *{ "\192\000\002\001", "\192\000\002\002" }*.
- **typeForAny** (*int*) – The type to use for raw responses when the requested type is `ANY`, as using `ANY` for the type of the response record would not make sense.

:suspend (*asyncID*, *queryID*, *timeoutMS*) → bool

New in version 1.8.0.

Suspend the processing for the current query, making it asynchronous. The query is then placed into memory, in a map called the Asynchronous Holder, until it is either resumed or the supplied timeout kicks in. The object is stored under a key composed of the tuple (*asyncID*, *queryID*) which is needed to retrieve it later, which can be done via `getAsynchronousObject()`. Note that the DNSQuestion object should NOT be accessed after successfully calling this method. Returns true on success and false on failure, indicating that the query has not been suspended and the normal processing will continue.

Parameters

- **asyncID** (*int*) – A numeric identifier used to identify the suspended query for later retrieval. Valid values range from 0 to 65535, both included.

- **queryID** (*int*) – A numeric identifier used to identify the suspended query for later retrieval. This ID does not have to match the query ID present in the initial DNS header. A given (asyncID, queryID) tuple should be unique at a given time. Valid values range from 0 to 65535, both included.
- **timeoutMS** (*int*) – The maximum duration this query will be kept in the asynchronous holder before being automatically resumed, in milliseconds.

19.10 DNSResponse object

class DNSResponse

This object has almost all the functions and members of a *DNSQuestion*, except for the following ones which are not available on a response:

- `addProxyProtocolValue`
- `ecsOverride`
- `ecsPrefixLength`
- `getProxyProtocolValues`
- `getHTTPHeaders`
- `getHTTPHost`
- `getHTTPPath`
- `getHTTPQueryString`
- `setHTTPResponse`
- `getHTTPScheme`
- `getServerNameIndication`
- `setNegativeAndAdditionalSOA`
- `setProxyProtocolValues`
- `spoof`
- `tempFailureTTL`
- `useECS`

If the value is really needed while the response is being processed, it is possible to set a tag while the query is processed, as tags will be passed to the response object. It also has additional methods:

getSelectedBackend() → Server

New in version 1.9.0.

Get the selected backend *`Server`* or nil

:editTTLs(*func*)

The function `func` is invoked for every entry in the answer, authority and additional section.

`func` points to a function with the following prototype: `myFunc(section, qclass, qtype, ttl)`

All parameters to `func` are integers:

- `section` is the section in the packet and can be compared to *DNS Packet Sections*
- `qclass` is the QClass of the record. Can be compared to *DNSClass*
- `qtype` is the QType of the record. Can be e.g. compared to `DNSQType.A`, `DNSQType.AAAA` *constants* and the like.
- `ttl` is the current TTL

This function must return an integer with the new TTL. Setting this TTL to 0 to leaves it unchanged

Parameters **func** (*string*) – The function to call to edit TTLs.

:changeName(*initialName*) → bool

New in version 1.8.0.

Change, in the DNS payload of the current response, the qname and the owner name of records to the supplied new name, if they are matching exactly the initial qname. This only makes if the reverse operation was performed on the query, or the client will be confused and likely drop the response. Note that only records whose owner name matches the qname in the initial response will be rewritten, and that only the owner name itself will be altered, not the content of the record rdata. For some records this might cause an issue with compression pointers contained in the payload, as they might no longer point to the correct position in the DNS payload. To prevent that, the records are checked against a list of supported record types, and the rewriting will not be performed if an unsupported type is present. As of 1.8.0 the list of supported types is: A, AAAA, DHCID, TXT, OPT, HINFO, DNSKEY, CDNSKEY, DS, CDS, DLV, SSHFP, KEY, CERT, TLSA, SMIMEA, OPENPGPKEY, NSEC, NSEC3, CSYNC, NSEC3PARAM, LOC, NID, L32, L64, EUI48, EUI64, URI, CAA, NS, PTR, CNAME, DNAME, RRSIG, MX, SOA, SRV Therefore this functionality only makes sense when the initial query is requesting a very simple type, like A or AAAA.

See also *DNSQuestion:changeName()*. Returns false on failure, true on success.

Parameters **initialName** (DNSName) – The initial qname

:restart()

New in version 1.8.0.

Discard the received response and restart the processing of the query. For this function to be usable, the query should have been made restartable first, via *DNSQuestion:setRestartable()*. For example, to restart the processing after selecting a different pool of servers:

```lua
function makeQueryRestartable(dq)
  -- make it possible to restart that query later
  -- by keeping a copy of the initial DNS payload around
  dq:setRestartable()
  return DNSAction.None
end
function restartOnServFail(dr)
  -- if the query was SERVFAIL and not already tried on the restarted pool
  if dr.rcode == DNSRCode.SERVFAIL and dr.pool ~= 'restarted' then
    -- assign this query to a new pool
    dr.pool = 'restarted'
    -- discard the received response and
    -- restart the processing of the query
    dr:restart()
  end
  return DNSResponseAction.None
end
addAction(AllRule(), LuaAction(makeQueryRestartable))
addResponseAction(AllRule(), LuaResponseAction(restartOnServFail))
```

19.11 DNSHeader (dh) object

class DNSHeader

This object holds a representation of a DNS packet's header.

:getAA() → bool

Get authoritative answer flag.

:getAD() → bool

Get authentic data flag.

:getCD() → bool
Get checking disabled flag.

:getID() → int
New in version 1.8.0.

Get the ID.

:getRA() → bool
Get recursion available flag.

:getRD() → bool
Get recursion desired flag.

:getTC() → bool
New in version 1.8.1.

Get the TC flag.

:setAA(*aa*)
Set authoritative answer flag.

Parameters **aa** (*bool*) – State of the AA flag

:setAD(*ad*)
Set authentic data flag.

Parameters **ad** (*bool*) – State of the AD flag

:setCD(*cd*)
Set checking disabled flag.

Parameters **cd** (*bool*) – State of the CD flag

:setQR(*qr*)
Set Query/Response flag. Setting QR to true means “This is an answer packet”.

Parameters **qr** (*bool*) – State of the QR flag

:setRA(*ra*)
Set recursion available flag.

Parameters **ra** (*bool*) – State of the RA flag

:setRD(*rd*)
Set recursion desired flag.

Parameters **rd** (*bool*) – State of the RD flag

:setTC(*tc*)
Set truncation flag (TC).

Parameters **tc** (*bool*) – State of the TC flag

19.12 EDNSOptionView object

class EDNSOptionView
An object that represents the values of a single EDNS option received in a query.

:count()
The number of values for this EDNS option.

:getValues()
Return a table of NULL-safe strings values for this EDNS option.

19.13 AsynchronousObject object

class AsynchronousObject

New in version 1.8.0.

This object holds a representation of a DNS query or response that has been suspended.

:drop() → bool

Drop that object immediately, without resuming it. Returns true on success, false on failure.

:getDQ() → DNSQuestion

Return a DNSQuestion object for the suspended object.

:getDR() → DNSResponse

Return a DNSResponse object for the suspended object.

:resume() → bool

Resume the processing of the suspended object. For a question, it means first checking whether it was turned into a response, and sending the response out it it was. Otherwise do a cache-lookup: on a cache-hit, the response will be sent immediately. On a cache-miss, it means dnsdist will select a backend and send the query to the backend. For a response, it means inserting into the cache if needed and sending the response to the backend. Note that the AsynchronousObject object should NOT be accessed after successfully calling this method. Returns true on success, false on failure.

:setRCode(*rcode*, *clearRecords*) → bool

Set the response code in the DNS header of the current object to the supplied value, optionally removing all records from the existing payload, if any. Returns true on success, false on failure.

Parameters

- **code** (`int`) – The response code to set
- **clearRecords** (`bool`) – Whether to clear all records from the existing payload, if any

getAsynchronousObject(*asyncID*, *queryID*) → AsynchronousObject

New in version 1.8.0.

Retrieves an asynchronous object stored into the Asynchronous holder.

param int asyncID A numeric identifier used to identify the query when it was suspended

param int queryID A numeric identifier used to identify the query when it was suspended

19.14 eBPF functions and objects

These are all the functions, objects and methods related to the *eBPF Socket Filtering*.

addBPFFilterDynBlocks(*addresses*, *dynbpf*[[, *seconds=10*], *msg*])

This is the eBPF equivalent of `addDynBlocks()`, blocking a set of addresses for (optionally) a number of seconds, using an eBPF dynamic filter. The default number of seconds to block for is 10. Since 1.6.0, the use of a *DynBlockRulesGroup* is a much more efficient way of doing the same thing.

Parameters

- **addresses** – set of Addresses as returned by an *exceed function*
- **dynbpf** (`DynBPFFilter`) – The dynamic eBPF filter to use
- **seconds** (`int`) – The number of seconds this block to expire
- **msg** (`str`) – A message to display while inserting the block

newBPFFilter(*options*) → BPFFilter

newBPFFilter(*v4Parameters*, *v6Parameters*, *qnamesParameters) -> BPFFilter (1.7.x*)
newBPFFilter(*maxV4*, *maxV6*, *maxQNames) -> BPFFilter (before 1.7.0*)

Changed in version 1.7.0: This function now supports a table for each parameters, and the ability to use pinned eBPF maps.

Changed in version 1.8.0: This function now gets its parameters via a table.

Return a new eBPF socket filter with a maximum of maxV4 IPv4, maxV6 IPv6 and maxQNames qname entries in the block tables. Maps can be pinned to a filesystem path, which makes their content persistent across restarts and allows external programs to read their content and to add new entries. dnsdist will try to load maps that are pinned to a filesystem path on startups, inheriting any existing entries, and fall back to creating them if they do not exist yet. Note that the user dnsdist is running under must have the right privileges to read and write to the given file, and to go through all the directories in the path leading to that file. The pinned path must be on a filesystem of type `BPF`, usually below `/sys/fs/bpf/`.

Parameters options (*table*) – A table with key: value pairs with options.

Options:

- `ipv4MaxItems`: int - The maximum number of entries in the IPv4 map. Default is 0 which will not allow any entry at all.
- `ipv4PinnedPath`: str - The filesystem path this map should be pinned to.
- `ipv6MaxItems`: int - The maximum number of entries in the IPv6 map. Default is 0 which will not allow any entry at all.
- `ipv6PinnedPath`: str - The filesystem path this map should be pinned to.
- `cidr4MaxItems`: int - The maximum number of entries in the IPv4 range block map. Default is 0 which will not allow any entry at all.
- `cidr4PinnedPath`: str - The filesystem path this map should be pinned to.
- `cidr6MaxItems`: int - The maximum number of entries in the IPv6 range block map. Default is 0 which will not allow any entry at all.
- `cidr6PinnedPath`: str - The filesystem path this map should be pinned to.
- `qnamesMaxItems`: int - The maximum number of entries in the qname map. Default is 0 which will not allow any entry at all.
- `qnamesPinnedPath`: str - The filesystem path this map should be pinned to.
- `external`: bool - If set to true, DNSDist does not load the internal eBPF program.

newDynBPFFilter(*bpf*) → DynBPFFilter

Return a new dynamic eBPF filter associated to a given BPF Filter.

Parameters bpf (`BPFFilter`) – The underlying eBPF filter

setDefaultBPFFilter(*filter*)

When used at configuration time, the corresponding BPFFilter will be attached to every bind.

Parameters filter (`BPFFilter`) – The filter to attach

registerDynBPFFilter(*dynbpf*)

Register a DynBPFFilter filter so that it appears in the web interface and the API.

Parameters dynbpf (`DynBPFFilter`) – The dynamic eBPF filter to register

unregisterDynBPFFilter(*dynbpf*)

Remove a DynBPFFilter filter from the web interface and the API.

Parameters dynbpf (`DynBPFFilter`) – The dynamic eBPF filter to unregister

class BPFFilter

Represents an eBPF filter

:attachToAllBinds()

Attach this filter to every bind already defined. This is the run-time equivalent of *setDefaultBPFFilter()*. This method can be used at run-time only.

:block(*address*)

Block this address

Parameters **address** (ComboAddress) – The address to block

:addRangeRule(*Netmask*, *action*[, *force=false*])

New in version 1.8.0.

Block all IP addresses in this range.

DNSDist eBPF code first checks if an exact IP match is found, then if a range matches, and finally if a DNSName does.

Parameters

- **Netmask** (*string*) – The ip range to block, allow or truncate
- **action** (*int*) – set `action` to `0` to allow a range, set `action` to `1` to block a range, set `action` to `2` to truncate a range.
- **force** (*bool*) – When `force` is set to true, DNSDist always accepts adding a new item to BPF maps, even if the item to be added may already be included in the larger network range.

:blockQName(*name*[, *qtype=255*])

Block queries for this exact qname. An optional qtype can be used, defaults to 255.

Parameters

- **name** (DNSName) – The name to block
- **qtype** (*int*) – QType to block

:getStats()

Print the block tables.

:unblock(*address*)

Unblock this address.

Parameters **address** (ComboAddress) – The address to unblock

:rmRangeRule(*Netmask*)

New in version 1.8.0.

Parameters **string** (Netmask) – The rule you want to remove

:lsRangeRule()

New in version 1.8.0.

List all range rule.

:unblockQName(*name*[, *qtype=255*])

Remove this qname from the block list.

Parameters

- **name** (DNSName) – the name to unblock
- **qtype** (*int*) – The qtype to unblock

class DynBPFFilter

Represents an dynamic eBPF filter, allowing the use of ephemeral rules to an existing eBPF filter. Note that since 1.6.0 the default BPF filter set via *setDefaultBPFFilter()* will automatically be used by a *DynBlockRulesGroup*, becoming the preferred way of dealing with ephemeral rules.

:purgeExpired()
Remove the expired ephemeral rules associated with this filter.

:excludeRange(*netmasks*)
Exclude this range, or list of ranges, meaning that no dynamic block will ever be inserted for clients in that range. Default to empty, meaning rules are applied to all ranges. When used in combination with `DynBPFFilter:includeRange()`, the more specific entry wins.

Parameters or list of str netmasks (*str*) – A netmask, or list of netmasks, as strings, like for example "192.0.2.1/24"

:includeRange(*netmasks*)
Include this range, or list of ranges, meaning that rules will be applied to this range. When used in combination with `DynBPFFilter:excludeRange()`, the more specific entry wins.

Parameters or list of str netmasks (*str*) – A netmask, or list of netmasks, as strings, like for example "192.0.2.1/24"

19.15 DNSCrypt objects and functions

addDNSCryptBind(*address*, *provider*, *certFile(s)*, *keyFile(s)*[, *options*])
Changed in version 1.4.0: Removed `doTCP` from the options. A listen socket on TCP is always created. `certFile(s)` and `keyFile(s)` now accept a list of files.

Changed in version 1.5.0: Added `tcpListenQueueSize` parameter.

Changed in version 1.6.0: Added `maxInFlight` and `maxConcurrentTCPConnections` parameters.

Adds a DNSCrypt listen socket on `address`.

Parameters

- **address** (*string*) – The address and port to listen on
- **provider** (*string*) – The provider name for this bind
- **certFile(s)** (*str*) – The path to a X.509 certificate file in PEM format, or a list of paths to such files.
- **keyFile(s)** (*str*) – The path to the private key file corresponding to the certificate, or a list of paths to such files, whose order should match the certFile(s) ones.
- **options** (*table*) – A table with key: value pairs with options (see below)

Options:

- `doTCP=true`: bool - Also bind on TCP on `address`, removed in 1.4.0.
- `reusePort=false`: bool - Set the `SO_REUSEPORT` socket option.
- `tcpFastOpenQueueSize=0`: int - Set the TCP Fast Open queue size, enabling TCP Fast Open when available and the value is larger than 0
- `interface=""`: str - Sets the network interface to use
- `cpus={}`: table - Set the CPU affinity for this listener thread, asking the scheduler to run it on a single CPU id, or a set of CPU ids. This parameter is only available if the OS provides the pthread_setaffinity_np() function.
- `tcpListenQueueSize=SOMAXCONN`: int - Set the size of the listen queue. Default is `SOMAXCONN`.
- `maxInFlight=0`: int - Maximum number of in-flight queries. The default is 0, which disables out-of-order processing.
- `maxConcurrentTCPConnections=0`: int - Maximum number of concurrent incoming TCP connections. The default is 0 which means unlimited.

generateDNSCryptProviderKeys (*publicKey*, *privateKey*)

Generate a new provider keypair and write them to `publicKey` and `privateKey`.

Parameters

- **publicKey** (*string*) – path to write the public key to
- **privateKey** (*string*) – path to write the private key to

generateDNSCryptCertificate (*privatekey*, *certificate*, *keyfile*, *serial*, *validFrom*, *validUntil*[, *version*])

generate a new resolver private key and related certificate, valid from the `validFrom` UNIX timestamp until the `validUntil` one, signed with the provider private key.

Parameters

- **privatekey** (*string*) – Path to the private key of the provider
- **certificate** (*string*) – Path where to write the certificate file
- **keyfile** (*string*) – Path where to write the private key for the certificate
- **serial** (*int*) – The certificate's serial number
- **validFrom** (*int*) – Unix timestamp from when the certificate will be valid
- **validUntil** (*int*) – Unix timestamp until when the certificate will be valid
- **version** (*DNSCryptExchangeVersion*) – The exchange version to use. Possible values are `DNSCryptExchangeVersion::VERSION1` (default, X25519-XSalsa20Poly1305) and `DNSCryptExchangeVersion::VERSION2` (X25519-XChacha20Poly1305)

printDNSCryptProviderFingerprint (*keyfile*)

Display the fingerprint of the provided resolver public key

Parameters **keyfile** (*string*) – Path to the key file

showDNSCryptBinds ()

Display the currently configured DNSCrypt binds

getDNSCryptBind (*n*) → DNSCryptContext

Return the *DNSCryptContext* object corresponding to the bind `n`.

getDNSCryptBindCount ()

New in version 1.5.0.

Return the number of DNSCrypt binds.

19.15.1 Certificates

class DNSCryptCert

Represents a DNSCrypt certificate.

:getClientMagic () → string

Return this certificate's client magic value.

:getEsVersion () → string

Return the cryptographic construction to use with this certificate,.

:getMagic () → string

Return the certificate magic number.

:getProtocolMinorVersion () → string

Return this certificate's minor version.

:getResolverPublicKey () → string

Return the public key corresponding to this certificate.

:getSerial() → int
Return the certificate serial number.

:getSignature() → string
Return this certificate's signature.

:getTSEnd() → int
Return the date the certificate is valid from, as a Unix timestamp.

:getTSStart() → int
Return the date the certificate is valid until (inclusive), as a Unix timestamp

19.15.2 Certificate Pairs

class DNSCryptCertificatePair
Represents a pair of DNSCrypt certificate and associated key

:getCertificate() → DNSCryptCert
Return the certificate.

:isActive() → bool
Return whether this pair is active and will be advertised to clients.

19.15.3 Context

class DNSCryptContext
Represents a DNSCrypt content. Can be used to rotate certs.

:addNewCertificate(*cert*, *key*[, *active*])
Add a new certificate to the given context. Active certificates are advertised to clients, inactive ones are not.

Parameters

- **cert** (`DNSCryptCert`) – The certificate to add to the context
- **key** (*`DNSCryptPrivateKey`*) – The private key corresponding to the certificate
- **active** (*`bool`*) – Whether the certificate should be advertised to clients. Default is true

:generateAndLoadInMemoryCertificate(*keyfile*, *serial*, *begin*, *end*[, *version*])
Generate a new resolver key and the associated certificate in-memory, sign it with the provided provider key, and add it to the context

Parameters

- **keyfile** (*`string`*) – Path to the provider key file to use
- **serial** (*`int`*) – The serial number of the certificate
- **begin** (*`int`*) – Unix timestamp from when the certificate is valid
- **end** (*`int`*) – Unix timestamp from until the certificate is valid
- **version** (*`DNSCryptExchangeVersion`*) – The exchange version to use. Possible values are `DNSCryptExchangeVersion::VERSION1` (default, X25519-XSalsa20Poly1305) and `DNSCryptExchangeVersion::VERSION2` (X25519-XChacha20Poly1305)

:getCertificate(*index*) → DNSCryptCert
Return the certificate with index *index*.

Parameters **index** (*`int`*) – The index of the certificate, starting at 0

:getCertificatePair(*index*) → DNSCryptCertificatePair

Return the certificate pair with index *index*.

Parameters **index** (*int*) – The index of the certificate, starting at 0

:getCertificatePair(*index*) → table of DNSCryptCertificatePair

Return a table of certificate pairs.

:getProviderName() → string

Return the provider name

:loadNewCertificate(*certificate*, *keyfile*[, *active*])

Load a new certificate and the corresponding private key. If *active* is false, the certificate will not be advertised to clients but can still be used to answer queries tied to it.

Parameters

- **certificate** (*string*) – Path to a certificate file
- **keyfile** (*string*) – Path to a the corresponding key file
- **active** (*bool*) – Whether the certificate should be marked as active. Default is true

:markActive(*serial*)

Mark the certificate with serial *serial* as active, meaning it will be advertised to clients.

Parameters **serial** (*int*) – The serial of the number to mark as active

:markInactive(*serial*)

Mark the certificate with serial *serial* as inactive, meaning it will not be advertised to clients but can still be used to answer queries tied to this certificate.

Parameters **serial** (*int*) – The serial of the number to mark as inactive

:printCertificates()

Print all the certificates.

:reloadCertificates()

New in version 1.6.0.

Reload the current TLS certificate and key pairs.

:removeInactiveCertificate(*serial*)

Remove the certificate with serial *serial*. It will not be possible to answer queries tied to this certificate, so it should have been marked as inactive for a certain time before that. Active certificates should be marked as inactive before they can be removed.

Parameters **serial** (*int*) – The serial of the number to remove

19.16 DNS Parser

Since 1.8.0, dnsdist contains a limited DNS parser class that can be used to inspect the content of DNS queries and responses in Lua.

The first step is to get the content of the DNS payload into a Lua string, for example using *DNSQuestion:getContent()*, or `DNSResponse:getContent()`, and then to create a *DNSPacketOverlay* object:

```
function dumpPacket(dq)
  local packet = dq:getContent()
  local overlay = newDNSPacketOverlay(packet)
  print(overlay.qname)
  print(overlay.qtype)
  print(overlay.qclass)
  local count = overlay:getRecordsCountInSection(DNSSection.Answer)
```

(continues on next page)

(continued from previous page)

```
  print(count)
  for idx=0, count-1 do
    local record = overlay:getRecord(idx)
    print(record.name)
    print(record.type)
    print(record.class)
    print(record.ttl)
    print(record.place)
    print(record.contentLength)
    print(record.contentOffset)
  end
  return DNSAction.None
end

addAction(AllRule(), LuaAction(dumpPacket))
```

newDNSPacketOverlay(*packet*) → DNSPacketOverlay

New in version 1.8.0.

Returns a DNSPacketOverlay

Parameters **packet** (*str*) – The DNS payload

19.16.1 DNSPacketOverlay

class DNSPacketOverlay

New in version 1.8.0.

The DNSPacketOverlay object has several attributes, all of them read-only:

qname

The qname of this packet, as a *DNSName objects*.

qtype

The type of the query in this packet.

qclass

The class of the query in this packet.

dh

It also supports the following methods:

:getRecordsCountInSection(*section*) → int

Returns the number of records in the ANSWER (1), AUTHORITY (2) and ADDITIONAL (3) *DNS Packet Sections* of this packet. The number of records in the QUESTION (0) is always set to 0, look at the dnsheader if you need the actual qdcount.

Parameters **section** (*int*) – The section, see above

:getRecord(*idx*) → DNSRecord

Get the record at the requested position. The records in the QUESTION sections are not taken into account, so the first record in the answer section would be at position 0.

Parameters **idx** (*int*) – The position of the requested record

19.17 DNSRecord object

class DNSRecord

New in version 1.8.0.

This object represents an unparsed DNS record, as returned by the *DNSPacketOverlay* class. It has several attributes, all of them read-only:

name
The name of this record, as a *DNSName objects*.

type
The type of this record.

class
The class of this record.

ttl
The TTL of this record.

place
The place (section) of this record.

contentLength
The length, in bytes, of the rdata content of this record.

contentOffset
The offset since the beginning of the DNS payload, in bytes, at which the rdata content of this record starts.

19.18 Protobuf Logging Reference

newRemoteLogger (*address*[, *timeout=2*[, *maxQueuedEntries=100*[, *reconnectWaitTime=1*]]])
Create a Remote Logger object, to use with `RemoteLogAction()` and `RemoteLogResponseAction()`.

Parameters

- **address** (*string*) – An IP:PORT combination where the logger is listening
- **timeout** (*int*) – TCP connect timeout in seconds
- **maxQueuedEntries** (*int*) – Queue this many messages before dropping new ones (e.g. when the remote listener closes the connection)
- **reconnectWaitTime** (*int*) – Time in seconds between reconnection attempts

class DNSDistProtoBufMessage
This object represents a single protobuf message as emitted by **dnsdist**.

:addResponseRR (*name*, *type*, *class*, *ttl*, *blob*)
Add a response RR to the protobuf message.

Parameters

- **name** (*string*) – The RR name.
- **type** (*int*) – The RR type.
- **class** (*int*) – The RR class.
- **ttl** (*int*) – The RR TTL.
- **blob** (*string*) – The RR binary content.

:setBytes (*bytes*)
Set the size of the query

Parameters bytes (*int*) – Number of bytes in the query.

:setEDNSSubnet (*netmask*)
Set the EDNS Subnet to `netmask`.

Parameters **netmask** (*string*) – The netmask to set to.

:setQueryTime(*sec*, *usec*)

In a response message, set the time at which the query has been received.

Parameters

- **sec** (*int*) – Unix timestamp when the query was received.
- **usec** (*int*) – The microsecond the query was received.

:setQuestion(*name*, *qtype*, *qclass*)

Set the question in the protobuf message.

Parameters

- **name** (DNSName) – The qname of the question
- **qtype** (*int*) – The qtype of the question
- **qclass** (*int*) – The qclass of the question

:setProtobufResponseType(*sec*, *usec*)

Change the protobuf response type from a query to a response, and optionally set the query time.

Parameters

- **sec** (*int*) – Optional query time in seconds.
- **usec** (*int*) – Optional query time in additional micro-seconds.

:setRequestor(*address*[, *port*])

Changed in version 1.5.0: `port` optional parameter added.

Set the requestor's address.

Parameters

- **address** (ComboAddress) – The address to set to
- **port** (*int*) – The requestor source port

:setRequestorFromString(*address*[, *port*])

Changed in version 1.5.0: `port` optional parameter added.

Set the requestor's address from a string.

Parameters

- **address** (*string*) – The address to set to
- **port** (*int*) – The requestor source port

:setResponder(*address*[, *port*])

Changed in version 1.5.0: `port` optional parameter added.

Set the responder's address.

Parameters

- **address** (ComboAddress) – The address to set to
- **port** (*int*) – The responder port

:setResponderFromString(*address*[, *port*])

Changed in version 1.5.0: `port` optional parameter added.

Set the responder's address.

Parameters

- **address** (*string*) – The address to set to
- **port** (*int*) – The responder port

:setResponseCode(*rcode*)
Set the response code of the query.

Parameters **rcode** (*int*) – The response code of the answer

:setServerIdentity(*id*)
Set the server identify field.

Parameters **id** (*string*) – The server ID

:setTag(*value*)
Add a tag to the list of tags.

Parameters **value** (*string*) – The tag value

:setTagArray(*valueList*)
Add a list of tags.

Parameters **tags** (*table*) – A list of tags as strings

:setTime(*sec*, *usec*)
Set the time at which the query or response has been received.

Parameters

- **sec** (*int*) – Unix timestamp when the query was received.
- **usec** (*int*) – The microsecond the query was received.

:toDebugString() → string
Return an string containing the content of the message

19.19 dnstap Logging Reference

dnstap is a flexible, structured binary log format for DNS software. Reader implementations in various languages exist.

dnsdist supports dnstap since version 1.3.0.

Canonically, dnstap is sent over a FrameStream socket, either a local AF_UNIX (see *newFrameStreamUnixLogger()*) or a TCP/IP socket (see *newFrameStreamTcpLogger()*). As an extension, **dnsdist** can send raw dnstap protobuf messages over a *newRemoteLogger()*.

To use FrameStream transport, **dnsdist** must have been built with *libfstrm*.

newFrameStreamUnixLogger(*path*[, *options*])
Changed in version 1.5.0: Added the optional parameter `options`.

Create a Frame Stream Logger object, to use with *DnstapLogAction()* and *DnstapLogResponseAction()*. This version will log to a local AF_UNIX socket.

Parameters

- **path** (*string*) – A local AF_UNIX socket path. Note that most platforms have a rather short limit on the length.
- **options** (*table*) – A table with key: value pairs with options.

The following options apply to the settings of the framestream library. Refer to the documentation of that library for the default and allowed values for these options, as well as their exact descriptions. For all these options, absence or a zero value has the effect of using the library-provided default value.

- `bufferHint=0`: unsigned
- `flushTimeout=0`: unsigned
- `inputQueueSize=0`: unsigned
- `outputQueueSize=0`: unsigned

- `queueNotifyThreshold=0`: unsigned
- `reopenInterval=0`: unsigned

`newFrameStreamTcpLogger`(*address*[, *options*])

Changed in version 1.5.0: Added the optional parameter `options`.

Create a Frame Stream Logger object, to use with *`DnstapLogAction()`* and *`DnstapLogResponseAction()`*. This version will log to a possibly remote TCP socket. Needs tcp_writer support in libfstrm.

Parameters

- **address** (*string*) – An IP:PORT combination where the logger will connect to.
- **options** (*table*) – A table with key: value pairs with options.

The following options apply to the settings of the framestream library. Refer to the documentation of that library for the default and allowed values for these options, as well as their exact descriptions. For all these options, absence or a zero value has the effect of using the library-provided default value.

- `bufferHint=0`: unsigned
- `flushTimeout=0`: unsigned
- `inputQueueSize=0`: unsigned
- `outputQueueSize=0`: unsigned
- `queueNotifyThreshold=0`: unsigned
- `reopenInterval=0`: unsigned

class DnstapMessage

This object represents a single dnstap message as emitted by **dnsdist**.

classmethod `DnstapMessage:`**`setExtra`**(*extraData*)

Sets the dnstap “extra” field.

Parameters **extraData** (*string*) – Extra data stuffed into the dnstap “extra” field.

classmethod `DnstapMessage:`**`toDebugString`**() → string

Return a string containing the content of the message

19.20 Carbon export

`carbonServer`(*serverIP*[, *ourname*[, *interval*[, *namespace*[, *instance*]]]])

Exort statistics to a Carbon / Graphite / Metronome server.

Parameters

- **serverIP** (*string*) – Indicates the IP address where the statistics should be sent
- **ourname** (*string*) – An optional string specifying the hostname that should be used
- **interval** (*int*) – An optional unsigned integer indicating the interval in seconds between exports
- **namespace** (*string*) – An optional string specifying the namespace name that should be used
- **instance** (*string*) – An optional string specifying the instance name that should be used

19.21 SNMP reporting

snmpAgent (*enableTraps*[, *daemonSocket*])
Enable SNMP support.

Parameters

- **enableTraps** (*bool*) – Indicates whether traps should be sent
- **daemonSocket** (*string*) – A string specifying how to connect to the daemon agent. This is a file path to a unix socket, but e.g. `tcp:localhost:705` can be used as well. By default, SNMP agent's default socket is used.

sendCustomTrap (*message*)
Send a custom SNMP trap from Lua.

Parameters **message** (*string*) – The message to include in the sent trap

19.22 Tuning related functions

setDoHDownstreamCleanupInterval (*interval*)
New in version 1.7.0.

Set how often, in seconds, the outgoing DoH connections to backends of a given worker thread are scanned to expunge the ones that are no longer usable. The default is 60 so once per minute and per worker thread.

Parameters **interval** (*int*) – The interval in seconds.

setDoHDownstreamMaxIdleTime (*max*)
New in version 1.7.0.

Set how long, in seconds, an outgoing DoH connection to a backend might stay idle before being closed. The default is 300 so 5 minutes.

Parameters **max** (*int*) – The maximum time in seconds.

setMaxIdleDoHConnectionsPerDownstream (*max*)
New in version 1.7.0.

Set the maximum number of inactive DoH connections to a backend cached by each DoH worker thread. These connections can be reused when a new query comes in, instead of having to establish a new connection. dnsdist regularly checks whether the other end has closed any cached connection, closing them in that case.

Parameters **max** (*int*) – The maximum number of inactive connections to keep. Default is 10, so 10 connections per backend and per DoH worker thread.

setMaxCachedTCPConnectionsPerDownstream (*max*)
New in version 1.6.0.

Set the maximum number of inactive TCP connections to a backend cached by each TCP worker thread. These connections can be reused when a new query comes in, instead of having to establish a new connection. dnsdist regularly checks whether the other end has closed any cached connection, closing them in that case.

Parameters **max** (*int*) – The maximum number of inactive connections to keep. Default is 10, so 10 connections per backend and per TCP worker thread.

setMaxTCPClientThreads (*num*)
Changed in version 1.6.0: Before 1.6.0 the default value was 10.

Changed in version 1.7.0: The default value has been set back to 10.

Set the maximum of TCP client threads, handling TCP connections. Before 1.4.0 a TCP thread could only handle a single incoming TCP connection at a time, while after 1.4.0 it can handle a larger number of them simultaneously.

Note that before 1.6.0 the TCP worker threads were created at runtime, adding a new thread when the existing ones seemed to struggle with the load, until the maximum number of threads had been reached. Starting with 1.6.0 the configured number of worker threads are immediately created at startup.

In 1.6.0 the default value was at least 10 TCP workers, but could be more if there is more than 10 TCP listeners (added via *`addDNSCryptBind()`*, *`addLocal()`*, or *`addTLSLocal()`*). In that last case there would have been as many TCP workers as TCP listeners. This led to issues in setups with a large number of TCP listeners and was therefore reverted back to 10 in 1.7.0.

Parameters **`num`** (*`int`*) – The number of TCP worker threads.

`setMaxTCPConnectionDuration` (*num*)

Set the maximum duration of an incoming TCP connection, in seconds. 0 (the default) means unlimited

Parameters **`num`** (*`int`*) –

`setMaxTCPConnectionsPerClient` (*num*)

Set the maximum number of TCP connections per client. 0 (the default) means unlimited

Parameters **`num`** (*`int`*) –

`setMaxTCPQueriesPerConnection` (*num*)

Set the maximum number of queries in an incoming TCP connection. 0 (the default) means unlimited

Parameters **`num`** (*`int`*) –

`setMaxTCPQueuedConnections` (*num*)

Changed in version 1.6.0: Before 1.6.0 the default value was 1000 on all systems.

Set the maximum number of TCP connections queued (waiting to be picked up by a client thread), defaults to 1000 (10000 on Linux since 1.6.0). 0 means unlimited

Parameters **`num`** (*`int`*) –

`setMaxUDPOutstanding` (*num*)

Changed in version 1.4.0: Before 1.4.0 the default value was 10240

Set the maximum number of outstanding UDP queries to a given backend server. This can only be set at configuration time and defaults to 65535 (10240 before 1.4.0)

Parameters **`num`** (*`int`*) –

`setCacheCleaningDelay` (*num*)

Set the interval in seconds between two runs of the cache cleaning algorithm, removing expired entries. Default is every 60s

Parameters **`num`** (*`int`*) –

`setCacheCleaningPercentage` (*num*)

Set the percentage of the cache that the cache cleaning algorithm will try to free by removing expired entries. By default (100), all expired entries are removed

Parameters **`num`** (*`int`*) –

`setOutgoingDoHWorkerThreads` (*num*)

New in version 1.7.0.

Set the number of worker threads to use for outgoing DoH. That number defaults to 0 but is automatically raised to 1 when DoH is enabled on at least one backend.

`setStaleCacheEntriesTTL` (*num*)

Allows using cache entries expired for at most n seconds when no backend available to answer for a query

Parameters **`num`** (*`int`*) –

setTCPDownstreamCleanupInterval(*interval*)
New in version 1.6.0.

Set how often, in seconds, the outgoing TCP connections to backends of a given worker thread are scanned to expunge the ones that are no longer usable. The default is 60 so once per minute and per worker thread.

Parameters **interval** (*int*) – The interval in seconds.

setDoHDownstreamMaxIdleTime(*max*)
New in version 1.7.0.

Set how long, in seconds, an outgoing DoH connection to a backend might stay idle before being closed. The default is 300 so 5 minutes.

Parameters **max** (*int*) – The maximum time in seconds.

setRandomizedIdsOverUDP(*val*)
New in version 1.8.0.

Setting this parameter to true (default is false) will randomize the IDs in outgoing UDP queries, at a small performance cost, ignoring the `setMaxUDPOutstanding()` value. This is only useful if the path between dnsdist and the backend is not trusted and the 'TCP-only', DNS over TLS or DNS over HTTPS transports cannot be used. See also `setRandomizedOutgoingSockets()`. The default is to use a linearly increasing counter from 0 to 65535, wrapping back to 0 when necessary.

setRandomizedOutgoingSockets(*val*)
New in version 1.8.0.

Setting this parameter to true (default is false) will randomize the outgoing socket used when forwarding a query to a backend. The default is to use a round-robin mechanism to select the outgoing socket. This requires configuring the backend to use more than one outgoing socket via the `sockets` parameter of `newServer()` to be of any use, and only makes sense if the path between dnsdist and the backend is not trusted and the 'TCP-only', DNS over TLS or DNS over HTTPS transports cannot be used. See also `setRandomizedIdsOverUDP()`.

setTCPInternalPipeBufferSize(*size*)
New in version 1.6.0.

Set the size in bytes of the internal buffer of the pipes used internally to distribute connections to TCP (and DoT) workers threads. Requires support for `F_SETPIPE_SZ` which is present in Linux since 2.6.35. The actual size might be rounded up to a multiple of a page size. 0 means that the OS default size is used. The default value is 0, except on Linux where it is 1048576 since 1.6.0.

Parameters **size** (*int*) – The size in bytes.

setTCPUseSinglePipe(*val*)
Deprecated since version 1.6.0.

Whether the incoming TCP connections should be put into a single queue instead of using per-thread queues. Defaults to false. That option was useful before 1.4.0 when a single TCP connection could block a TCP worker thread, but should not be used in recent versions where the per-thread queues model avoids waking up all idle workers when a new connection arrives. This option will be removed in 1.7.0.

Parameters **val** (*bool*) –

setTCPRecvTimeout(*num*)
Set the read timeout on TCP connections from the client, in seconds. Defaults to 2

Parameters **num** (*int*) –

setTCPSendTimeout(*num*)
Set the write timeout on TCP connections from the client, in seconds. Defaults to 2

Parameters **num** (*int*) –

setUDPMultipleMessagesVectorSize(*num*)
Set the maximum number of UDP queries messages to accept in a single `recvmmsg()` call. Only available

if the underlying OS support `recvmmsg()` with the `MSG_WAITFORONE` option. Defaults to 1, which means only query at a time is accepted, using `recvmsg()` instead of `recvmmsg()`.

Parameters **num** (*int*) – maximum number of UDP queries to accept

setUDPSocketBufferSizes(*recv*, *send*)

New in version 1.7.0.

Set the size of the receive (`SO_RCVBUF`) and send (`SO_SNDBUF`) buffers for incoming UDP sockets. On Linux the default values correspond to `net.core.rmem_default` and `net.core.wmem_default` , and the maximum values are restricted by `net.core.rmem_max` and `net.core.wmem_max`. Since 1.9.0, on Linux, dnsdist will automatically try to raise the buffer sizes to the maximum value allowed by the system (`net.core.rmem_max` and `net.core.wmem_max`) if *setUDPSocketBufferSizes()* is not set.

Parameters

- **recv** (*int*) – `SO_RCVBUF` value. Default is 0, meaning the system value will be kept.
- **send** (*int*) – `SO_SNDBUF` value. Default is 0, meaning the system value will be kept.

setUDPTimeout(*num*)

Set the maximum time dnsdist will wait for a response from a backend over UDP, in seconds. Defaults to 2

Parameters **num** (*int*) –

19.23 Key Value Store functions and objects

These are all the functions, objects and methods related to the CDB and LMDB key value stores.

A lookup into a key value store can be done via the *KeyValueStoreLookupRule()* rule or the *KeyValueStoreLookupAction()* action, using the usual selectors to match the incoming queries for which the lookup should be done.

The first step is to get a *KeyValueStore* object via one of the following functions:

- *newCDBKVStore()* for a CDB database ;
- *newLMDBKVStore()* for a LMDB one.

Then the key used for the lookup can be selected via one of the following functions:

- the exact qname with *KeyValueLookupKeyQName()* ;
- a suffix match via *KeyValueLookupKeySuffix()*, meaning that several lookups will be done, removing one label from the qname at a time, until a match has been found or there is no label left ;
- the source IP, in network byte order, with *KeyValueLookupKeySourceIP()* ;
- the value of an existing tag with *KeyValueLookupKeyTag()*.

For example, to do a suffix-based lookup into a LMDB KVS database, the following rule can be used:

```
> kvs = newLMDBKVStore('/path/to/lmdb/database', 'database name')
> addAction(AllRule(), KeyValueStoreLookupAction(kvs, KeyValueLookupKeySuffix(),
↪'kvs-suffix-result'))
```

For a query whose qname is “sub.domain.powerdns.com.”, and for which only the “\8powerdns\3com\0” key exists in the database, this would result in the following lookups:

- \3sub\6domain\8powerdns\3com\0
- \6domain\8powerdns\3com\0
- \8powerdns\3com\0

Then a match is found for the last key, and the corresponding value is stored into the 'kvs-suffix-result' tag. This tag can now be used in subsequent rules to take an action based on the result of the lookup. Note that the tag is also created when the key has not been found, but the content of the tag is empty.

```
> addAction(TagRule('kvs-suffix-result', 'this is the value obtained from the␣
↪lookup'), SpoofAction('2001:db8::1'))
```

If the value found in the LMDB database for the key '\8powerdns\3com\0' was 'this is the value obtained from the lookup', then the query is immediately answered with a AAAA record.

class KeyValueStore

New in version 1.4.0.

Represents a Key Value Store

:lookup(*key*[, *wireFormat*])

Does a lookup into the corresponding key value store, and return the result as a string. The key can be a *ComboAddress* obtained via the *newCA()*, a *DNSName* obtained via the *newDNSName()* function, or a raw string.

Parameters

- **DNSName or string key** (*ComboAddress,*) – The key to look up
- **wireFormat** (*bool*) – If the key is DNSName, whether to use to do the lookup in wire format (default) or in plain text

:lookupSuffix(*key*[, *minLabels*[, *wireFormat*]])

Does a suffix-based lookup into the corresponding key value store, and return the result as a string. The key should be a *DNSName* object obtained via the *newDNSName()* function, and several lookups will be done, removing one label from the name at a time until a match has been found or there is no label left. If `minLabels` is set to a value larger than 0 the lookup will only be done as long as there is at least `minLabels` remaining. For example if the initial domain is "sub.powerdns.com." and `minLabels` is set to 2, lookups will only be done for "sub.powerdns.com." and "powerdns.com.".

Parameters

- **key** (DNSName) – The name to look up
- **minLabels** (*int*) – The minimum number of labels to do a lookup for. Default is 0 which means unlimited
- **wireFormat** (*bool*) – Whether to do the lookup in wire format (default) or in plain text

:reload()

Reload the database if this is supported by the underlying store. As of 1.4.0, only CDB stores can be reloaded, and this method is a no-op for LMDB stores.

KeyValueLookupKeyQName([*wireFormat*]) → KeyValueLookupKey

New in version 1.4.0.

Return a new KeyValueLookupKey object that, when passed to *KeyValueStoreLookupAction()* or *KeyValueStoreLookupRule()*, will return the qname of the query in DNS wire format.

Parameters wireFormat (*bool*) – Whether to do the lookup in wire format (default) or in plain text

KeyValueLookupKeySourceIP([*v4mask*[, *v6mask*]]) → KeyValueLookupKey

New in version 1.4.0.

Changed in version 1.5.0: Optional parameters `v4mask` and `v6mask` added.

Changed in version 1.7.0: Optional parameter `includePort` added.

Return a new KeyValueLookupKey object that, when passed to *KeyValueStoreLookupAction()* or *KeyValueStoreLookupRule()*, will return the source IP of the client in network byte-order.

Parameters

- **v4mask** (*int*) – Mask applied to IPv4 addresses. Default is 32 (the whole address)
- **v6mask** (*int*) – Mask applied to IPv6 addresses. Default is 128 (the whole address)
- **includePort** (*int*) – Whether to append the port (in network byte-order) after the address. Default is false

KeyValueLookupKeySuffix([*minLabels*[, *wireFormat*]]) → KeyValueLookupKey

New in version 1.4.0.

Return a new KeyValueLookupKey object that, when passed to `KeyValueStoreLookupAction()` or `KeyValueStoreLookupRule()`, will return a vector of keys based on the labels of the qname in DNS wire format or plain text. For example if the qname is sub.domain.powerdns.com. the following keys will be returned:

- \3sub\6domain\8powerdns\3com\0
- \6domain\8powerdns\3com\0
- \8powerdns\3com\0
- \3com\0
- \0

If `minLabels` is set to a value larger than 0 the lookup will only be done as long as there is at least `minLabels` remaining. Taking back our previous example, it means only the following keys will be returned if `minLabels` is set to 2;

- \3sub\6domain\8powerdns\3com\0
- \6domain\8powerdns\3com\0
- \8powerdns\3com\0

Parameters

- **minLabels** (*int*) – The minimum number of labels to do a lookup for. Default is 0 which means unlimited
- **wireFormat** (*bool*) – Whether to do the lookup in wire format (default) or in plain text

KeyValueLookupKeyTag(*tagName*) → KeyValueLookupKey

New in version 1.4.0.

Return a new KeyValueLookupKey object that, when passed to `KeyValueStoreLookupAction()`, will return the value of the corresponding tag for this query, if it exists.

Parameters **tagName** (*str*) – The name of the tag.

newCDBKVStore(*filename*, *refreshDelay*) → KeyValueStore

New in version 1.4.0.

Return a new KeyValueStore object associated to the corresponding CDB database. The modification time of the CDB file will be checked every 'refreshDelay' second and the database re-opened if needed.

Parameters

- **filename** (*string*) – The path to an existing CDB database
- **refreshDelays** (*int*) – The delay in seconds between two checks of the database modification time. 0 means disabled

newLMDBKVStore(*filename*, *dbName*[, *noLock*]) → KeyValueStore

New in version 1.4.0.

Changed in version 1.7.0: Added the optional parameter `noLock`.

Return a new KeyValueStore object associated to the corresponding LMDB database. The database must have been created with the `MDB_NOSUBDIR` flag. Since 1.7.0, the database is opened with the `MDB_READONLY` flag, and optionally with `MDB_NOLOCK` if `noLock` is set to true.

Parameters

- **`filename`** (*`string`*) – The path to an existing LMDB database created with `MDB_NOSUBDIR`
- **`dbName`** (*`string`*) – The name of the database to use
- **`noLock`** (*`bool`*) – Whether to open the database with the `MDB_NOLOCK` flag. Default is false

19.24 Logging

There are some functions to create log output.

`errlog`(*line*)

Writes a error line.

Parameters **`line`** (*`str`*) – The line to write.

`warnlog`(*line*)

Writes a warning line.

Parameters **`line`** (*`str`*) – The line to write.

`infolog`(*line*)

Writes an info line.

Parameters **`line`** (*`str`*) – The line to write.

`vinfolog`(*line*)

New in version 1.8.0.

Writes an info line if dnsdist is running in verbose (debug) mode.

Parameters **`line`** (*`str`*) – The line to write.

19.25 Webserver-related objects

`class WebRequest`

Represent a HTTP query, whose attributes are read-only.

`body`

The body of this query, as a string.

`getvars`

The GET parameters of this query, as a table whose keys and values are strings.

`headers`

The HTTP headers of this query, as a table whose keys and values are strings.

`method`

The method of this query, as a string.

`path`

The path of this query, as a string.

`postvars`

The POST parameters of this query, as a table whose keys and values are strings.

`version`

The HTTP version of this query, as an integer.

class WebResponse

Represent a HTTP response.

body

The body of this response, as a string.

headers

The HTTP headers of this response, as a table whose keys and values are strings.

status

The HTTP status code of this response, as an integer.

19.26 Rules management

19.26.1 Incoming queries

For Rules related to the incoming query:

addAction (*DNSrule*, *action*[, *options*])

Changed in version 1.6.0: Added `name` to the `options`.

Changed in version 1.9.0: Passing a string or list of strings instead of a `DNSRule` is deprecated, use `NetmaskGroupRule()` or `QNameSuffixRule()` instead

Add a Rule and Action to the existing rules. If a string (or list of) is passed as the first parameter instead of a `DNSRule`, it behaves as if the string or list of strings was passed to `NetmaskGroupRule()` or `SuffixMatchNodeRule()`.

Parameters

- **rule** (*DNSrule*) – A `DNSRule`, e.g. an `AllRule()`, or a compounded bunch of rules using e.g. `AndRule()`. Before 1.9.0 it was also possible to pass a string (or list of strings) but doing so is now deprecated.
- **action** – The action to take
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `uuid`: string - UUID to assign to the new rule. By default a random UUID is generated for each rule.
- `name`: string - Name to assign to the new rule.

clearRules ()

Remove all current rules.

getAction (*n*) → DNSDistRuleAction

Returns the `DNSDistRuleAction` associated with rule `n`.

Parameters n (*int*) – The rule number

getRule (*selector*) → DNSDistRuleAction

New in version 1.9.0.

Return the rule corresponding to the selector, if any. The selector can be the position of the rule in the list, as an integer, its name as a string or its UUID as a string as well.

Parameters or str selector (*int*) – The position in the list, name or UUID of the rule to return.

mvRule (*from*, *to*)

Move rule `from` to a position where it is in front of `to`. `to` can be one larger than the largest rule, in which case the rule will be moved to the last position.

Parameters

- **from** (*int*) – Rule number to move
- **to** (*int*) – Location to more the Rule to

mvRuleToTop()

New in version 1.6.0.

This function moves the last rule to the first position. Before 1.6.0 this was handled by `topRule()`.

setRules(*rules*)

Replace the current rules with the supplied list of pairs of DNS Rules and DNS Actions (see `newRuleAction()`)

Parameters **rules** (*[RuleAction]*) – A list of RuleActions

showRules([*options*])

Show all defined rules for queries, optionally displaying their UUIDs.

Parameters **options** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.
- `truncateRuleWidth=-1`: int - Truncate rules output to `truncateRuleWidth` size. Defaults to `-1` to display the full rule.

topRule()

Changed in version 1.6.0: Replaced by `mvRuleToTop()`

Before 1.6.0 this function used to move the last rule to the first position, which is now handled by `mvRuleToTop()`.

rmRule(*id*)

Changed in version 1.6.0: `id` can now be a string representing the name of the rule.

Remove rule `id`.

Parameters **id** (*int*) – The position of the rule to remove if `id` is numerical, its UUID or name otherwise

19.26.2 Cache misses

For Rules related to the incoming query after a cache miss:

> **Warning:** While all selectors and actions are available, some actions will no longer be honored at this point. For example changing the backend pool will not trigger a second cache-lookup. Switching from a backend pool that has EDNS Client Subnet enabled to one that doesn't will result in the EDNS Client Subnet corresponding to the initial server pool to be added to the query.

addCacheMissAction(*DNSrule*, *action*[, *options*])

New in version 1.10.

Add a Rule and Action to the existing cache miss rules. If a string (or list of) is passed as the first parameter instead of a `DNSRule`, it behaves as if the string or list of strings was passed to `NetmaskGroupRule()` or `SuffixMatchNodeRule()`.

Parameters

- **rule** (*DNSrule*) – A `DNSRule`, e.g. an `AllRule()`, or a compounded bunch of rules using e.g. `AndRule()`.
- **action** – The action to take
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `uuid`: string - UUID to assign to the new rule. By default a random UUID is generated for each rule.
- `name`: string - Name to assign to the new rule.

clearCacheMissRules()

New in version 1.10.

Remove all current cache miss rules.

getCacheMissAction(*n*) → DNSDistRuleAction

New in version 1.10.

Returns the `DNSDistRuleAction` associated with cache miss rule `n`.

Parameters **n** (`int`) – The rule number

getCacheMissRule(*selector*) → DNSDistRuleAction

New in version 1.10.

Return the cache miss rule corresponding to the selector, if any. The selector can be the position of the rule in the list, as an integer, its name as a string or its UUID as a string as well.

Parameters **or str selector** (`int`) – The position in the list, name or UUID of the rule to return.

mvCacheMissRule(*from*, *to*)

New in version 1.10.

Move cache miss rule `from` to a position where it is in front of `to`. `to` can be one larger than the largest rule, in which case the rule will be moved to the last position.

Parameters

- **from** (`int`) – Rule number to move
- **to** (`int`) – Location to more the Rule to

mvCacheMissRuleToTop()

New in version 1.10.

This function moves the last cache miss rule to the first position.

setCacheMissRules(*rules*)

New in version 1.10.

Replace the current cache miss rules with the supplied list of pairs of DNS Rules and DNS Actions (see `newRuleAction()`)

Parameters **rules** (`[RuleAction]`) – A list of RuleActions

showCacheMissRules([*options*])

New in version 1.10.

Show all defined cache miss rules for queries, optionally displaying their UUIDs.

Parameters **options** (`table`) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.
- `truncateRuleWidth=-1`: int - Truncate rules output to `truncateRuleWidth` size. Defaults to `-1` to display the full rule.

rmCacheMissRule(*id*)

New in version 1.10.

Remove rule `id`.

Parameters **id** (`int`) – The position of the cache miss rule to remove if `id` is numerical, its UUID or name otherwise

19.26.3 Responses

For Rules related to responses:

`addResponseAction` (*DNSRule*, *action*[, *options*])

Changed in version 1.6.0: Added `name` to the `options`.

Changed in version 1.9.0: Passing a string or list of strings instead of a `DNSRule` is deprecated, use `NetmaskGroupRule()` or `QNameSuffixRule()` instead

Add a Rule and Action for responses to the existing rules. If a string (or list of) is passed as the first parameter instead of a `DNSRule`, it behaves as if the string or list of strings was passed to `NetmaskGroupRule()` or `SuffixMatchNodeRule()`.

Parameters

- **`rule`** (*`DNSrule`*) – A `DNSRule`, e.g. an `AllRule()`, or a compounded bunch of rules using e.g. `AndRule()`. Before 1.9.0 it was also possible to pass a string (or list of strings) but doing so is now deprecated.
- **`action`** – The action to take
- **`options`** (*`table`*) – A table with key: value pairs with options.

Options:

- `uuid`: string - UUID to assign to the new rule. By default a random UUID is generated for each rule.
- `name`: string - Name to assign to the new rule.

`clearResponseRules` ()

New in version 1.10.

Remove all current response rules.

`getResponseRule` (*selector*) → DNSDistResponseRuleAction

New in version 1.9.0.

Return the response rule corresponding to the selector, if any. The selector can be the position of the rule in the list, as an integer, its name as a string or its UUID as a string as well.

Parameters or str selector (*`int`*) – The position in the list, name or UUID of the rule to return.

`mvResponseRule` (*from*, *to*)

Move response rule `from` to a position where it is in front of `to`. `to` can be one larger than the largest rule, in which case the rule will be moved to the last position.

Parameters

- **`from`** (*`int`*) – Rule number to move
- **`to`** (*`int`*) – Location to more the Rule to

`mvResponseRuleToTop` ()

New in version 1.6.0.

This function moves the last response rule to the first position. Before 1.6.0 this was handled by `topResponseRule()`.

`rmResponseRule` (*id*)

Changed in version 1.6.0: `id` can now be a string representing the name of the rule.

Remove response rule `id`.

Parameters id (*`int`*) – The position of the rule to remove if `id` is numerical, its UUID or name otherwise

`showResponseRules` ([*options*])

Show all defined response rules, optionally displaying their UUIDs.

Parameters options (*table*) – A table with key: value pairs with display options.

Options:

- showUUIDs=false: bool - Whether to display the UUIDs, defaults to false.
- truncateRuleWidth=-1: int - Truncate rules output to truncateRuleWidth size. Defaults to -1 to display the full rule.

topResponseRule()

Changed in version 1.6.0: Replaced by *mvResponseRuleToTop()*

Before 1.6.0 this function used to move the last response rule to the first position, which is now handled by *mvResponseRuleToTop()*.

19.26.4 Cache hits

Functions for manipulating Cache Hit Response Rules:

addCacheHitResponseAction(*DNSRule*, *action*[, *options*])

Changed in version 1.6.0: Added name to the options.

Changed in version 1.9.0: Passing a string or list of strings instead of a *DNSRule* is deprecated, use *NetmaskGroupRule()* or *QNameSuffixRule()* instead

Add a Rule and ResponseAction for Cache Hits to the existing rules. If a string (or list of) is passed as the first parameter instead of a *DNSRule*, it behaves as if the string or list of strings was passed to *NetmaskGroupRule()* or *SuffixMatchNodeRule()*.

Parameters

- **rule** (*DNSrule*) – A *DNSRule*, e.g. an *AllRule()*, or a compounded bunch of rules using e.g. *AndRule()*. Before 1.9.0 it was also possible to pass a string (or list of strings) but doing so is now deprecated.
- **action** – The action to take
- **options** (*table*) – A table with key: value pairs with options.

Options:

- uuid: string - UUID to assign to the new rule. By default a random UUID is generated for each rule.
- name: string - Name to assign to the new rule.

clearCacheHitResponseRules()

New in version 1.10.

Remove all current cache-hit response rules.

getCacheHitResponseRule(*selector*) → DNSDistResponseRuleAction

New in version 1.9.0.

Return the cache-hit response rule corresponding to the selector, if any. The selector can be the position of the rule in the list, as an integer, its name as a string or its UUID as a string as well.

Parameters or str selector (*int*) – The position in the list, name or UUID of the rule to return.

mvCacheHitResponseRule(*from*, *to*)

Move cache hit response rule from to a position where it is in front of to. to can be one larger than the largest rule, in which case the rule will be moved to the last position.

Parameters

- **from** (*int*) – Rule number to move
- **to** (*int*) – Location to more the Rule to

mvCacheHitResponseRuleToTop()

New in version 1.6.0.

This function moves the last cache hit response rule to the first position. Before 1.6.0 this was handled by *topCacheHitResponseRule()*.

rmCacheHitResponseRule(*id*)

Changed in version 1.6.0: `id` can now be a string representing the name of the rule.

Parameters **id** (*int*) – The position of the rule to remove if `id` is numerical, its UUID or name otherwise

showCacheHitResponseRules([*options*])

Show all defined cache hit response rules, optionally displaying their UUIDs.

Parameters **options** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.
- `truncateRuleWidth=-1`: int - Truncate rules output to `truncateRuleWidth` size. Defaults to `-1` to display the full rule.

topCacheHitResponseRule()

Changed in version 1.6.0: Replaced by *mvCacheHitResponseRuleToTop()*

Before 1.6.0 this function used to move the last cache hit response rule to the first position, which is now handled by *mvCacheHitResponseRuleToTop()*.

19.26.5 Cache inserted

Functions for manipulating Cache Inserted Response Rules:

addCacheInsertedResponseAction(*DNSRule*, *action*[, *options*])

New in version 1.8.0.

Changed in version 1.9.0: Passing a string or list of strings instead of a *DNSRule* is deprecated, use *NetmaskGroupRule()* or *QNameSuffixRule()* instead

Add a Rule and ResponseAction that is executed after a cache entry has been inserted to the existing rules. If a string (or list of) is passed as the first parameter instead of a *DNSRule*, it behaves as if the string or list of strings was passed to *NetmaskGroupRule()* or *SuffixMatchNodeRule()*.

Parameters

- **rule** (*DNSrule*) – A *DNSRule*, e.g. an *AllRule()*, or a compounded bunch of rules using e.g. *AndRule()*. Before 1.9.0 it was also possible to pass a string (or list of strings) but doing so is now deprecated.
- **action** – The action to take
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `uuid`: string - UUID to assign to the new rule. By default a random UUID is generated for each rule.
- `name`: string - Name to assign to the new rule.

clearCacheInsertedResponseRules()

New in version 1.10.

Remove all current cache-inserted response rules.

getCacheInsertedResponseRule(*selector*) → DNSDistResponseRuleAction

New in version 1.9.0.

Return the cache-inserted response rule corresponding to the selector, if any. The selector can be the position of the rule in the list, as an integer, its name as a string or its UUID as a string as well.

Parameters **`or str selector`** (*`int`*) – The position in the list, name or UUID of the rule to return.

`mvCacheInsertedResponseRule`(*from*, *to*)

New in version 1.8.0.

Move cache inserted response rule `from` to a position where it is in front of `to`. `to` can be one larger than the largest rule, in which case the rule will be moved to the last position.

Parameters

- **`from`** (*`int`*) – Rule number to move
- **`to`** (*`int`*) – Location to more the Rule to

`mvCacheInsertedResponseRuleToTop`()

New in version 1.8.0.

This function moves the last cache inserted response rule to the first position.

`rmCacheInsertedResponseRule`(*id*)

New in version 1.8.0.

Parameters **`id`** (*`int`*) – The position of the rule to remove if `id` is numerical, its UUID or name otherwise

`showCacheInsertedResponseRules`([*options*])

New in version 1.8.0.

Show all defined cache inserted response rules, optionally displaying their UUIDs.

Parameters **`options`** (*`table`*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.
- `truncateRuleWidth=-1`: int - Truncate rules output to `truncateRuleWidth` size. Defaults to `-1` to display the full rule.

19.26.6 Self-answered responses

Functions for manipulating Self-Answered Response Rules:

`addSelfAnsweredResponseAction`(*DNSRule*, *action*[, *options*])

Changed in version 1.6.0: Added `name` to the `options`.

Changed in version 1.9.0: Passing a string or list of strings instead of a *`DNSRule`* is deprecated, use *`NetmaskGroupRule()`* or *`QNameSuffixRule()`* instead

Add a Rule and Action for Self-Answered queries to the existing rules. If a string (or list of) is passed as the first parameter instead of a *`DNSRule`*, it behaves as if the string or list of strings was passed to *`NetmaskGroupRule()`* or *`SuffixMatchNodeRule()`*.

Parameters

- **`rule`** (*`DNSrule`*) – A *`DNSRule`*, e.g. an *`AllRule()`*, or a compounded bunch of rules using e.g. *`AndRule()`*. Before 1.9.0 it was also possible to pass a string (or list of strings) but doing so is now deprecated.
- **`action`** – The action to take
- **`options`** (*`table`*) – A table with key: value pairs with options.

Options:

- `uuid`: string - UUID to assign to the new rule. By default a random UUID is generated for each rule.

- `name`: string - Name to assign to the new rule.

`clearSelfAnsweredResponseRules`()

New in version 1.10.

Remove all current self-answered response rules.

`getSelfAnsweredResponseRule`(*selector*) → DNSDistResponseRuleAction

New in version 1.9.0.

Return the self-answered response rule corresponding to the selector, if any. The selector can be the position of the rule in the list, as an integer, its name as a string or its UUID as a string as well.

Parameters **`or str selector`** (*int*) – The position in the list, name or UUID of the rule to return.

`mvSelfAnsweredResponseRule`(*from*, *to*)

Move self answered response rule `from` to a position where it is in front of `to`. `to` can be one larger than the largest rule, in which case the rule will be moved to the last position.

Parameters

- **`from`** (*int*) – Rule number to move
- **`to`** (*int*) – Location to more the Rule to

`mvSelfAnsweredResponseRuleToTop`()

New in version 1.6.0.

This function moves the last self-answered response rule to the first position. Before 1.6.0 this was handled by *`topSelfAnsweredResponseRule()`*.

`rmSelfAnsweredResponseRule`(*id*)

Changed in version 1.6.0: `id` can now be a string representing the name of the rule.

Remove self answered response rule `id`.

Parameters **`id`** (*int*) – The position of the rule to remove if `id` is numerical, its UUID or name otherwise

`showSelfAnsweredResponseRules`([*options*])

Show all defined self answered response rules, optionally displaying their UUIDs.

Parameters **`options`** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.
- `truncateRuleWidth=-1`: int - Truncate rules output to `truncateRuleWidth` size. Defaults to `-1` to display the full rule.

`topSelfAnsweredResponseRule`()

Changed in version 1.6.0: Replaced by *`mvSelfAnsweredResponseRuleToTop()`*

Before 1.6.0 this function used to move the last self-answered response rule to the first position, which is now handled by *`mvSelfAnsweredResponseRuleToTop()`*.

Move the last self answered response rule to the first position.

19.26.7 XFR

Functions for manipulating zone transfer (AXFR, IXFR) Response Rules:

Note: Please remember that a zone transfer (XFR) can and will often contain several response packets to a single query packet.

> **Warning:** While almost all existing selectors and Response actions should be usable from the XFR response rules, it is strongly advised to only inspect the content of XFR response packets, and not modify them. Logging the content of response packets can be done via:
>
> - `DnstapLogResponseAction()`
> - `LogResponseAction()`
> - `RemoteLogResponseAction()`

addXFRResponseAction (*DNSRule*, *action*[, *options*])

New in version 1.10.

Add a Rule and ResponseAction for zone transfers (XFR) to the existing rules. If a string (or list of) is passed as the first parameter instead of a `DNSRule`, it behaves as if the string or list of strings was passed to `NetmaskGroupRule()` or `SuffixMatchNodeRule()`.

Parameters

- **rule** (*DNSrule*) – A `DNSRule`, e.g. an `AllRule()`, or a compounded bunch of rules using e.g. `AndRule()`.
- **action** – The action to take
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `uuid`: string - UUID to assign to the new rule. By default a random UUID is generated for each rule.
- `name`: string - Name to assign to the new rule.

mvXFRResponseRule (*from*, *to*)

New in version 1.10.

Move XFR response rule `from` to a position where it is in front of `to`. `to` can be one larger than the largest rule, in which case the rule will be moved to the last position.

Parameters

- **from** (*int*) – Rule number to move
- **to** (*int*) – Location to more the Rule to

mvXFRResponseRuleToTop ()

New in version 1.10.

This function moves the last XFR response rule to the first position.

rmXFRResponseRule (*id*)

New in version 1.10.

Parameters **id** (*int*) – The position of the rule to remove if `id` is numerical, its UUID or name otherwise

showXFRResponseRules ([*options*])

New in version 1.10.

Show all defined XFR response rules, optionally displaying their UUIDs.

Parameters **options** (*table*) – A table with key: value pairs with display options.

Options:

- `showUUIDs=false`: bool - Whether to display the UUIDs, defaults to false.
- `truncateRuleWidth=-1`: int - Truncate rules output to `truncateRuleWidth` size. Defaults to `-1` to display the full rule.

19.26.8 Convenience Functions

makeRule(*rule*)

Changed in version 1.9.0: This function is deprecated, please use *NetmaskGroupRule()* or QnameSuffixRule() instead

Make a *NetmaskGroupRule()* or a *SuffixMatchNodeRule()*, depending on how it is called. The *rule* parameter can be a string, or a list of strings, that should contain either:

- netmasks: in which case it will behave as *NetmaskGroupRule()*, or
- domain names: in which case it will behave as *SuffixMatchNodeRule()*

Mixing both netmasks and domain names is not supported, and will result in domain names being ignored!

`makeRule("0.0.0.0/0")` will for example match all IPv4 traffic, `makeRule({"be","nl","lu"})` will match all Benelux DNS traffic.

Parameters **rule** (*string*) – A string, or list of strings, to convert to a rule.

newRuleAction(*rule*, *action*[, *options*])

Changed in version 1.6.0: Added `name` to the `options`.

Return a pair of DNS Rule and DNS Action, to be used with *setRules()*.

Parameters

- **rule** (*Rule*) – A Rule (see *Rule selectors*)
- **action** (*Action*) – The Action (see *Rule Actions*) to apply to the matched traffic
- **options** (*table*) – A table with key: value pairs with options.

Options:

- `uuid`: string - UUID to assign to the new rule. By default a random UUID is generated for each rule.
- `name`: string - Name to assign to the new rule.

19.27 Rule selectors

Packets can be matched by selectors, called a `DNSRule`.

These `DNSRules` be one of the following items:

- A string that is either a domain name or netmask
- A list of strings that are either domain names or netmasks
- A *DNSName*
- A list of *DNSName*s
- A (compounded) `Rule`

Selectors can be combined via *AndRule()*, *OrRule()* and *NotRule()*.

AllRule()

Matches all traffic

DNSSECRule()

Matches queries with the DO flag set

DSTPortRule(*port*)

Matches questions received to the destination port.

Parameters **port** (*int*) – Match destination port.

EDNSOptionRule(*optcode*)
New in version 1.4.0.

Matches queries or responses with the specified EDNS option present. `optcode` is specified as an integer, or a constant such as *EDNSOptionCode.ECS*.

EDNSVersionRule(*version*)
New in version 1.4.0.

Matches queries or responses with an OPT record whose EDNS version is greater than the specified EDNS version.

Parameters **version** (`int`) – The EDNS version to match on

ERCodeRule(*rcode*)
Matches queries or responses with the specified `rcode`. `rcode` can be specified as an integer or as one of the built-in *RCode*. The full 16bit RCode will be matched. If no EDNS OPT RR is present, the upper 12 bits are treated as 0.

Parameters **rcode** (`int`) – The RCODE to match on

HTTPHeaderRule(*name*, *regex*)
New in version 1.4.0.

Changed in version 1.8.0: see `keepIncomingHeaders` on `addDOHLocal()`

Matches DNS over HTTPS queries with a HTTP header `name` whose content matches the regular expression `regex`. Since 1.8.0 it is necessary to set the `keepIncomingHeaders` option to true on `addDOHLocal()` to be able to use this rule.

Parameters

- **name** (`str`) – The case-insensitive name of the HTTP header to match on
- **regex** (`str`) – A regular expression to match the content of the specified header

HTTPPathRegexRule(*regex*)
New in version 1.4.0.

Matches DNS over HTTPS queries with a HTTP path matching the regular expression supplied in `regex`. For example, if the query has been sent to the https://192.0.2.1:443/PowerDNS?dns=… URL, the path would be '/PowerDNS'. Only valid DNS over HTTPS queries are matched. If you want to match all HTTP queries, see `DOHFrontend:setResponsesMap()` instead.

Parameters **regex** (`str`) – The regex to match on

HTTPPathRule(*path*)
New in version 1.4.0.

Matches DNS over HTTPS queries with a HTTP path of `path`. For example, if the query has been sent to the https://192.0.2.1:443/PowerDNS?dns=… URL, the path would be '/PowerDNS'. Only valid DNS over HTTPS queries are matched. If you want to match all HTTP queries, see `DOHFrontend:setResponsesMap()` instead.

Parameters **path** (`str`) – The exact HTTP path to match on

KeyValueStoreLookupRule(*kvs*, *lookupKey*)
New in version 1.4.0.

Return true if the key returned by 'lookupKey' exists in the key value store referenced by 'kvs'. The store can be a CDB (`newCDBKVStore()`) or a LMDB database (`newLMDBKVStore()`). The key can be based on the qname (`KeyValueLookupKeyQName()` and `KeyValueLookupKeySuffix()`), source IP (`KeyValueLookupKeySourceIP()`) or the value of an existing tag (`KeyValueLookupKeyTag()`).

Parameters

- **kvs** (`KeyValueStore`) – The key value store to query
- **lookupKey** (`KeyValueLookupKey`) – The key to use for the lookup

KeyValueStoreRangeLookupRule (*kvs*, *lookupKey*)

New in version 1.7.0.

Does a range-based lookup into the key value store referenced by 'kvs' using the key returned by 'lookupKey' and returns true if there is a range covering that key.

This assumes that there is a key, in network byte order, for the last element of the range (for example 2001:0db8:ffff:ffff:ffff:ffff:ffff:ffff for 2001:db8::/32) which contains the first element of the range (2001:0db8:0000:0000:0000:0000:0000:0000) (optionally followed by any data) as value, still in network byte order, and that there is no overlapping ranges in the database. This requires that the underlying store supports ordered keys, which is true for LMDB but not for CDB.

Parameters

- **kvs** (`KeyValueStore`) – The key value store to query
- **lookupKey** (*KeyValueLookupKey*) – The key to use for the lookup

LuaFFIPerThreadRule (*function*)

New in version 1.7.0.

Invoke a Lua FFI function that accepts a pointer to a `dnsdist_ffi_dnsquestion_t` object, whose bindings are defined in `dnsdist-lua-ffi.hh`.

The `function` should return true if the query matches, or false otherwise. If the Lua code fails, false is returned.

The function will be invoked in a per-thread Lua state, without access to the global Lua state. All constants (*DNSQType*, *RCode*, …) are available in that per-thread context, as well as all FFI functions. Objects and their bindings that are not usable in a FFI context (*DNSQuestion*, *DNSDistProtoBufMessage*, *PacketCache*, …) are not available.

Parameters **function** (*string*) – a Lua string returning a Lua function

LuaFFIRule (*function*)

New in version 1.5.0.

Invoke a Lua FFI function that accepts a pointer to a `dnsdist_ffi_dnsquestion_t` object, whose bindings are defined in `dnsdist-lua-ffi.hh`.

The `function` should return true if the query matches, or false otherwise. If the Lua code fails, false is returned.

Parameters **function** (*string*) – the name of a Lua function

LuaRule (*function*)

New in version 1.5.0.

Invoke a Lua function that accepts a *DNSQuestion* object.

The `function` should return true if the query matches, or false otherwise. If the Lua code fails, false is returned.

Parameters **function** (*string*) – the name of a Lua function

MaxQPSIPRule (*qps*[, *v4Mask*[, *v6Mask*[, *burst*[, *expiration*[, *cleanupDelay*[, *scanFraction*[, *shards*]]]]]]])

Changed in version 1.8.0: `shards` parameter added

Matches traffic for a subnet specified by `v4Mask` or `v6Mask` exceeding `qps` queries per second up to `burst` allowed. This rule keeps track of QPS by netmask or source IP. This state is cleaned up regularly if `cleanupDelay` is greater than zero, removing existing netmasks or IP addresses that have not been seen in the last `expiration` seconds.

Parameters

- **qps** (*int*) – The number of queries per second allowed, above this number traffic is matched
- **v4Mask** (*int*) – The IPv4 netmask to match on. Default is 32 (the whole address)

- **v6Mask** (*int*) – The IPv6 netmask to match on. Default is 64
- **burst** (*int*) – The number of burstable queries per second allowed. Default is same as qps
- **expiration** (*int*) – How long to keep netmask or IP addresses after they have last been seen, in seconds. Default is 300
- **cleanupDelay** (*int*) – The number of seconds between two cleanups. Default is 60
- **scanFraction** (*int*) – The maximum fraction of the store to scan for expired entries, for example 5 would scan at most 20% of it. Default is 10 so 10%
- **shards** (*int*) – How many shards to use, to decrease lock contention between threads. Default is 10 and is a safe default unless a very high number of threads are used to process incoming queries

MaxQPSRule (*qps*)

Matches traffic **not** exceeding this qps limit. If e.g. this is set to 50, starting at the 51st query of the current second traffic stops being matched. This can be used to enforce a global QPS limit.

Parameters **qps** (*int*) – The number of queries per second allowed, above this number the traffic is **not** matched anymore

NetmaskGroupRule (*nmg*[, *src*[, *quiet*]])

Changed in version 1.4.0: `quiet` parameter added

Changed in version 1.9.0: The `nmg` parameter now accepts a string or a list of strings in addition to a class:*NetmaskGroup* object.

Matches traffic from/to the network range specified in the `nmg`, which can be a string, a list of strings, or a *NetmaskGroup* object created via *newNMG()*.

Set the `src` parameter to false to match `nmg` against destination address instead of source address. This can be used to differentiate between clients

Parameters

- **nmg** (NetmaskGroup) – The netmasks to match, can be a string, a list of strings or a *NetmaskGroup* object.
- **src** (*bool*) – Whether to match source or destination address of the packet. Defaults to true (matches source)
- **quiet** (*bool*) – Do not display the list of matched netmasks in Rules. Default is false.

OpcodeRule (*code*)

Matches queries with opcode `code`. `code` can be directly specified as an integer, or one of the *built-in DNSOpcodes*.

Parameters **code** (*int*) – The opcode to match

PayloadSizeRule (*comparison*, *size*)

New in version 1.9.0.

Matches queries or responses whose DNS payload size fits the given comparison.

Parameters

- **comparison** (*str*) – The comparison operator to use. Supported values are `equal`, `greater`, `greaterOrEqual`, `smaller` and `smallerOrEqual`.
- **size** (*int*) – The size to compare to.

ProbaRule (*probability*)

Matches queries with a given probability. 1.0 means "always"

Parameters **probability** (*double*) – Probability of a match

ProxyProtocolValueRule(*type*[, *value*])

New in version 1.6.0.

Matches queries that have a proxy protocol TLV value of the specified type. If `value` is set, the content of the value should also match the content of `value`.

Parameters

- **type** (*int*) – The type of the value, ranging from 0 to 255 (both included)
- **value** (*str*) – The optional binary-safe value to match

QClassRule(*qclass*)

Matches queries with the specified `qclass`. `class` can be specified as an integer or as one of the built-in *DNSClass*.

Parameters **qclass** (*int*) – The Query Class to match on

QNameRule(*qname*)

Matches queries with the specified qname exactly.

Parameters **qname** (*string*) – Qname to match

QNameSetRule(*set*)

New in version 1.4.0: Matches if the set contains exact qname.

To match subdomain names, see `QNameSuffixRule()`.

param DNSNameSet set Set with qnames of type class:*DNSNameSet* created with `newDNSNameSet()`.

QNameSuffixRule(*suffixes*[, *quiet*])

New in version 1.9.0.

Matches based on a group of domain suffixes for rapid testing of membership. The first parameter, `suffixes`, can be a string, list of strings or a class:*SuffixMatchNode* object created with `newSuffixMatchNode()`. Pass true as second parameter to prevent listing of all domains matched.

To match domain names exactly, see `QNameSetRule()`.

This rule existed before 1.9.0 but was called `SuffixMatchNodeRule()`, only accepting a `SuffixMatchNode` parameter.

Parameters

- **suffixes** – A string, list of strings, or a `SuffixMatchNode` to match on
- **quiet** (*bool*) – Do not display the list of matched domains in Rules. Default is false.

  Matches queries with the specified qname exactly.

  param string qname Qname to match

QNameLabelsCountRule(*min*, *max*)

Matches if the qname has less than `min` or more than `max` labels.

Parameters

- **min** (*int*) – Minimum number of labels
- **max** (*int*) – Maximum nimber of labels

QNameWireLengthRule(*min*, *max*)

Matches if the qname's length on the wire is less than `min` or more than `max` bytes.

Parameters

- **min** (*int*) – Minimum number of bytes
- **max** (*int*) – Maximum nimber of bytes

QTypeRule (*qtype*)

Matches queries with the specified `qtype` `qtype` may be specified as an integer or as one of the built-in QTypes. For instance `DNSQType.A`, `DNSQType.TXT` and `DNSQType.ANY`.

Parameters **qtype** (*int*) – The QType to match on

RCodeRule (*rcode*)

Matches queries or responses with the specified `rcode`. `rcode` can be specified as an integer or as one of the built-in *RCode*. Only the non-extended RCode is matched (lower 4bits).

Parameters **rcode** (*int*) – The RCODE to match on

RDRule ()

Matches queries with the RD flag set.

RegexRule (*regex*)

Matches the query name against the `regex`.

```
addAction(RegexRule("[0-9]{5,}"), DelayAction(750)) -- milliseconds
addAction(RegexRule("[0-9]{4,}\\.example$"), DropAction())
```

This delays any query for a domain name with 5 or more consecutive digits in it. The second rule drops anything with more than 4 consecutive digits within a .EXAMPLE domain.

Note that the query name is presented without a trailing dot to the regex. The regex is applied case insensitively.

Parameters **regex** (*string*) – A regular expression to match the traffic on

RecordsCountRule (*section*, *minCount*, *maxCount*)

Matches if there is at least `minCount` and at most `maxCount` records in the section `section`. `section` can be specified as an integer or as a *DNS Packet Sections*.

Parameters

- **section** (*int*) – The section to match on
- **minCount** (*int*) – The minimum number of entries
- **maxCount** (*int*) – The maximum number of entries

RecordsTypeCountRule (*section*, *qtype*, *minCount*, *maxCount*)

Matches if there is at least `minCount` and at most `maxCount` records of type `type` in the section `section`. `section` can be specified as an integer or as a *DNS Packet Sections*. `qtype` may be specified as an integer or as one of the *built-in QTypes*, for instance `DNSQType.A` or `DNSQType.TXT`.

Parameters

- **section** (*int*) – The section to match on
- **qtype** (*int*) – The QTYPE to match on
- **minCount** (*int*) – The minimum number of entries
- **maxCount** (*int*) – The maximum number of entries

RE2Rule (*regex*)

Matches the query name against the supplied regex using the RE2 engine.

For an example of usage, see *RegexRule()*.

Note Only available when **dnsdist** was built with libre2 support.

Parameters **regex** (*str*) – The regular expression to match the QNAME.

SNIRule (*name*)

New in version 1.4.0.

Matches against the TLS Server Name Indication value sent by the client, if any. Only makes sense for DoT or DoH, and for that last one matching on the HTTP Host header using *HTTPHeaderRule()* might provide more consistent results. As of the version 2.3.0-beta of h2o, it is unfortunately not possible to extract the SNI value from DoH connections, and it is therefore necessary to use the HTTP Host header until version 2.3.0 is released, or `nghttp2` is used for incoming DoH instead (1.9.0+).

Parameters **name** (*str*) – The exact SNI name to match.

SuffixMatchNodeRule(*smn*[, *quiet*])

Changed in version 1.9.0: The `smn` parameter now accepts a string or a list of strings in addition to a class:*SuffixMatchNode* object.

Matches based on a group of domain suffixes for rapid testing of membership. The first parameter, `smn`, can be a string, list of strings or a class:*SuffixMatchNode* object created with `newSuffixMatchNode()`. Pass true as second parameter to prevent listing of all domains matched.

To match domain names exactly, see `QNameSetRule()`.

Since 1.9.0, this rule can also be used via the alias `QNameSuffixRule()`.

Parameters

- **smn** (`SuffixMatchNode`) – A string, list of strings, or a `SuffixMatchNode` to match on
- **quiet** (*bool*) – Do not display the list of matched domains in Rules. Default is false.

TagRule(*name*[, *value*])

Matches question or answer with a tag named `name` set. If `value` is specified, the existing tag value should match too.

Parameters

- **name** (*string*) – The name of the tag that has to be set
- **value** (*string*) – If set, the value the tag has to be set to. Default is unset

TCPRule(*tcp*)

Matches question received over TCP if `tcp` is true, over UDP otherwise.

Parameters **tcp** (*bool*) – Match TCP traffic if true, UDP traffic if false.

TrailingDataRule()

Matches if the query has trailing data.

PoolAvailableRule(*poolname*)

Check whether a pool has any servers available to handle queries

```
--- Send queries to default pool when servers are available
addAction(PoolAvailableRule(""), PoolAction(""))
--- Send queries to fallback pool if not
addAction(AllRule(), PoolAction("fallback"))
```

Parameters **poolname** (*string*) – Pool to check

PoolOutstandingRule(*poolname*, *limit*)

New in version 1.7.0.

Check whether a pool has total outstanding queries above limit

```
--- Send queries to spill over pool if default pool is under pressure
addAction(PoolOutstandingRule("", 5000), PoolAction("spillover"))
```

Parameters

- **poolname** (*string*) – Pool to check
- **limit** (*int*) – Total outstanding limit

19.27.1 Combining Rules

AndRule (*selectors*)

Matches traffic if all `selectors` match.

Parameters **selectors** (*{Rule}*) – A table of Rules

NotRule (*selector*)

Matches the traffic if the `selector` rule does not match;

Parameters **selector** (*Rule*) – A Rule

OrRule (*selectors*)

Matches the traffic if one or more of the `selectors` Rules does match.

Parameters **selector** (*{Rule}*) – A table of Rules

19.27.2 Objects

class DNSDistRuleAction

New in version 1.9.0.

Represents a rule composed of a *DNSRule* selector, to select the queries this applies to, and a `DNSAction` action to apply when the selector matches.

:getAction ()

Return the `DNSAction` action of this rule.

:getSelector ()

Return the *DNSRule* selector of this rule.

class DNSDistResponseRuleAction

New in version 1.9.0.

Represents a rule composed of a *DNSRule* selector, to select the responses this applies to, and a `DNSResponseAction` action to apply when the selector matches.

:getAction ()

Return the `DNSResponseAction` action of this rule.

:getSelector ()

Return the *DNSRule* selector of this rule.

class DNSRule

New in version 1.9.0.

:getMatches () → int

Return the number of times this selector matched a query or a response. Note that if the same selector is reused for different `DNSDistRuleAction` objects, the counter will be common to all these objects.

19.28 SVCRecordParameters

newSVCRecordParameters (*priority*, *target*[, *SVCParams*]) → SVCRecordParameters

New in version 1.7.0.

Returns a *SVCRecordParameters* to use with *SpoofSVCAction()*.

```
-- reply to SVCB queries for _dns.resolver.arpa. indicating DoT on port 853 of␣
↪dot.powerdns.com. (192.0.2.1/2001:db8::1), DoH on https://doh.powerdns.com/
↪dns-query (192.0.2.2/2001:db8::2)
local svc = { newSVCRecordParameters(1, "dot.powerdns.com.", { mandatory={"port
↪"}, alpn={ "dot" }, noDefaultAlpn=true, port=853, ipv4hint={ "192.0.2.1" },␣
↪ipv6hint={ "2001:db8::1" } }),
```

(continues on next page)

(continued from previous page)

```
               newSVCRecordParameters(2, "doh.powerdns.com.", { mandatory={"port
↪"}, alpn={ "h2" }, port=443, ipv4hint={ "192.0.2.2" }, ipv6hint={
↪"2001:db8::2" }, key7 = "/dns-query{?dns}" })
             }
addAction(AndRule{QTypeRule(64), QNameRule('_dns.resolver.arpa.')},␣
↪SpoofSVCAction(svc))
-- reply with NODATA (NXDOMAIN would deny all types at that name and below,␣
↪including SVC) for other types
addAction(QNameRule('_dns.resolver.arpa.'), NegativeAndSOAAction(false, '_dns.
↪resolver.arpa.', 3600, 'fake.resolver.arpa.', 'fake.resolver.arpa.', 1, 1800,
↪ 900, 604800, 86400))
```

Parameters

- **priority** (*int*) – The priority of this record. if more than one record is returned, they all should have different priorities. A priority of 0 indicates Alias mode and no other record should be present in the RRSet.
- **target** (*str*) – A domain name indicating the target name.
- **SVCParams** (*table*) – Optional table of additionals parameters. The key should be the name of the SVC parameter and will be used as the SvcParamKey, while the value depends on the key (see below)

These SVCParams can be set:

```
{
  mandatory={STRING},    -- The mandatory keys. the table of strings must be␣
↪the key names (like "port" and "key998").
  alpn={STRING},         -- alpns for this record, like "dot" or "h2".
  noDefaultAlpn=BOOL,    -- When true, the no-default-alpn key is included in␣
↪the record, false or absent means it does not exist in the record.
  port=NUM,              -- Port parameter to include.
  ipv4hint={STRING},     -- IPv4 hints to include into the record.
  ech=STRING,            -- Encrypted Client Hello as a raw string (can include␣
↪null bytes).
  ipv6hint={STRING}      -- IPv6 hints to include into the record.
}
```

Any other parameters can be set by using the `keyNNNN` syntax and must use a raw string. Like this:

```
key776="hello\0world"
```

class SVCRecordParameters

New in version 1.7.0.

Represents Service Binding (SVCB, HTTPS) record parameters, which can be used with *SpoofSVCAction()*.

19.29 Custom Metrics

You can define your own metrics that can be updated using Lua.

The first step is to declare a new metric using *declareMetric()*. In 1.8.0 the declaration had to be done at configuration time, but since 1.8.1 it can be done at any point.

Then you can update those at runtime using the following functions, depending on the metric type:

- manipulate counters using *incMetric()* and *decMetric()*
- update a gauge using *setMetric()*

declareMetric(*name*, *type*, *description*[, *prometheusName*]) → bool

New in version 1.8.0.

Changed in version 1.8.1: This function can now be used at runtime, instead of only at configuration time.

Return true if declaration was successful

Parameters

- **name** (*str*) – The name of the metric, lowercase alphanumerical characters and dashes (-) only
- **type** (*str*) – The desired type in `gauge` or `counter`
- **name** – The description of the metric
- **prometheusName** (*str*) – The name to use in the prometheus metrics, if supplied. Otherwise the regular name will be used, prefixed with `dnsdist_` and `-` replaced by _.

incMetric(*name*[, *step*]) → int

New in version 1.8.0.

Changed in version 1.8.1: Optional `step` parameter added.

Increment counter by one (or more, see the `step` parameter), will issue an error if the metric is not declared or not a `counter` Return the new value

Parameters

- **name** (*str*) – The name of the metric
- **step** (*int*) – By how much the counter should be incremented, default to 1.

decMetric(*name*) → int

New in version 1.8.0.

Changed in version 1.8.1: Optional `step` parameter added.

Decrement counter by one (or more, see the `step` parameter), will issue an error if the metric is not declared or not a `counter` Return the new value

Parameters

- **name** (*str*) – The name of the metric
- **step** (*int*) – By how much the counter should be decremented, default to 1.

getMetric(*name*) → double

New in version 1.8.0.

Get metric value

Parameters **name** (*str*) – The name of the metric

setMetric(*name*, *value*) → double

New in version 1.8.0.

Set the new value, will issue an error if the metric is not declared or not a `gauge` Return the new value

Parameters

- **name** (*str*) – The name of the metric
- **value** (*double*) – The new value

19.30 XSK / AF_XDP functions and objects

These are all the functions, objects and methods related to *AF_XDP / XSK*.

newXSK(*options*)

New in version 1.9.0.

This function creates a new *XskSocket* object, tied to a network interface and queue, to accept XSK / AF_XDP packet from the Linux kernel. The returned object can be passed as a parameter to *addLocal()* to use XSK for UDP packets between clients and dnsdist. It can also be passed to newServer to use XSK for UDP packets between dnsdist a backend.

Parameters **options** (*table*) – A table with key: value pairs with listen options.

Options:

- ifName: str - The name of the network interface this object will be tied to.
- NIC_queue_id: int - The queue of the network interface this object will be tied to.
- frameNums: int - The number of UMEM frames to allocate for this socket. More frames mean that a higher number of packets can be processed at the same time. 65535 is a good choice for maximum performance.
- xskMapPath: str - The path of the BPF map used to communicate with the kernel space XDP program, usually /sys/fs/bpf/dnsdist/xskmap.

class XskSocket

New in version 1.9.0.

Represents a XSK / AF_XDP socket tied to a specific network interface and queue. This object can be created via :func:newXSK and passed to *addLocal()* to use XSK for UDP packets between clients and dnsdist. It can also be passed to newServer to use XSK for UDP packets between dnsdist a backend.

:getMetrics() → str

Returns a string containing XSK / AF_XDP metrics for this object, as reported by the Linux kernel.

CHAPTER

TWENTY

MANUAL PAGES

20.1 dnsdist

20.1.1 Synopsis

dnsdist [<option>. . .] [address]. . .

20.1.2 Description

dnsdist receives DNS queries and relays them to one or more downstream servers. It subsequently sends back responses to the original requestor.

dnsdist operates over TCP and UDP, and strives to deliver very high performance over both.

Currently, queries are sent to the downstream server with the least outstanding queries. This effectively implies load balancing, making sure that slower servers get less queries.

If a reply has not come in after a few seconds, it is removed from the queue, but in the short term, timeouts do cause a server to get less traffic.

IPv4 and IPv6 operation can be mixed and matched, in other words, queries coming in over IPv6 could be forwarded to IPv4 and vice versa.

dnsdist is scriptable in Lua, see the dnsdist documentation for more information on this.

20.1.3 Scope

dnsdist does not 'think' about DNS queries, it restricts itself to measuring response times and error codes and routing questions accordingly. It comes with a very high performance packet-cache.

The goal for dnsdist is to remain simple. If more powerful loadbalancing is required, dedicated hardware or software is recommended. Linux Virtual Server for example is often mentioned.

20.1.4 Options

-a <netmask>, --acl <netmask> Add *netmask* to the ACL.

-C <file>, --config <file> Load configuration from *file*.

--check-config Test the configuration file (which may be set with **–config** or **-C**) for errors. dnsdist will show the errors and exit with a non-zero exit-code when errors are found.

-c <address>, --client <address> Operate as a client, connect to dnsdist. This will read the dnsdist configuration for the **controlSocket** statement and connect to it. When *address* (with an optional port number) is set, dnsdist will connect to that instead.

-k <key>, --setkey <key> When operating as a client(**-c**, **–client**), use *key* as shared secret to connect to dnsdist. This should be the same key that is used on the server (set with **setKey()**). Note that this will leak the key into your shell's history and into the systems running process list. Only available when dnsdist is compiled with libsodium or libcrypto support.

-e, --execute <command> Connect to dnsdist and execute *command*.

-h, --help Display a helpful message and exit.

-l, --local <address> Bind to *address*, Supply as many addresses (using multiple **–local** statements) to listen on as required. Specify IPv4 as 0.0.0.0:53 and IPv6 as [::]:53.

--supervised Run in foreground, but do not spawn a console. Use this switch to run dnsdist inside a supervisor (use with e.g. systemd and daemontools).

--disable-syslog Disable logging to syslog. Use this when running inside a supervisor that handles logging (like systemd).

--log-timestamps Prepend timestamps to messages logged to standard out.

-u, --uid <uid> Change the process user to *uid* after binding sockets. *uid* can be a name or number.

-g, --gid <gid> Change the process group to *gid* after binding sockets. *gid* Can be a name or number.

-V, --version Show the dnsdist version and exit.

-v, --verbose Be verbose.

address are any number of downstream DNS servers, in the same syntax as used with **–local**. If the port is not specified, 53 is used.

20.1.5 Bugs

Right now, the TCP support has some rather arbitrary limits.

20.1.6 Resources

Website: https://dnsdist.org

CHAPTER

TWENTYONE

CHANGELOG

21.1 1.9.3

Released: 5th of April 2024

21.1.1 Bug Fixes

- Revert “Release failed TCP backend connections more quickly” to fix a crash ¶ References: pull request 14040

21.2 1.9.2

Released: 5th of April 2024

21.2.1 Improvements

- Fix compilation warnings ¶ References: pull request 13938
- Docker: Only print config if debug flag is set ¶ References: pull request 13939
- Shrink InternalQueryState’s size by reordering its fields ¶ References: pull request 13942
- Fix annoying compiler warnings by introducing and switching to *pdns::UniqueFilePtr* ¶ References: #13925, pull request 13943
- Support “no server available” result from Lua FFI load-balancing policies ¶ References: #13977, pull request 14013
- Release incoming TCP connection right away on backend failure ¶ References: pull request 14016
- Release failed TCP backend connections more quickly ¶ References: pull request 14017

21.2.2 Bug Fixes

- Use server preference algorithm for ALPN selection ¶ References: #13850, pull request 13940
- Fix a null-deref in incoming DNS over HTTPS with the nghttp2 provider ¶ References: pull request 14012
- Fix DNS over HTTP connections/queries counters with the *nghttp2* provider ¶ References: pull request 14014
- Fix first IPv6 console connection being rejected ¶ References: #13903, pull request 13941
- Fix XSK-enabled check when reconnecting a backend ¶ References: pull request 13944
- Properly handle a failure of the first lazy health-check ¶ References: #13837, pull request 13945

- Also handle EHOSTUNREACH as a case for reconnecting the socket ¶ References: #13945, pull request 13976
- FDWrapper: Do not try to close negative file descriptors ¶ References: pull request 14015

21.3 1.9.1

Released: 14th of March 2024

This release does not contain any dnsdist code changes compared to 1.9.0. The only thing that changed is the version of Quiche, because of a security update.

Please review the *Upgrade Guide* before upgrading.

21.3.1 Bug Fixes

- update Quiche to 0.20.1. Fixes CVE-2024-1410 and CVE-2024-1765. ¶ References: pull request 13912

21.4 1.9.0

Released: 16th of February 2024

Please review the *Upgrade Guide* before upgrading.

21.4.1 Improvements

- Better handling of short, non-initial QUIC headers ¶ References: pull request 13755
- Fix performance inefficiencies reported by Coverity ¶ References: pull request 13779
- Fix a warning reported by Coverity ¶ References: pull request 13757
- Add a Lua maintenance hook ¶ References: pull request 13768

21.4.2 Bug Fixes

- Fix a missing explicit atomic load of the Quiche configuration ¶ References: pull request 13774
- Do not allocate 16-byte aligned objects through lua(jit) ¶ References: #13766, pull request 13771

21.5 1.9.0-rc1

Released: 30th of January 2024

Please review the *Upgrade Guide* before upgrading.

21.5.1 New Features

- Add AF_XDP support for UDP (Y7n05h) ¶ References: pull request 11652

21.5.2 Improvements

- Enable DoQ and DoH3 in dockerfile-dnsdist (Denis Machard) ¶ References: pull request 13674
- Enable PMTU discovery and disable fragmentation on QUIC binds ¶ References: pull request 13676
- Fall back to libcrypto for authenticated encryption ¶ References: pull request 13650
- Optimize the DoQ packet handling path ¶ References: pull request 13666
- Increase UDP receive and send buffers to the maximum allowed ¶ References: pull request 13664
- Clean up the Lua objects before exiting ¶ References: pull request 13667
- Cleanup of code doing SNMP OID handling ¶ References: pull request 13711
- Fix missed optimizations reported by Coverity ¶ References: pull request 13727
- Move the console socket instead of copying it ¶ References: pull request 13735
- DNSName: Correct len and offset types ¶ References: pull request 13723
- DNSName: Optimize parsing of uncompressed labels ¶ References: pull request 13724

21.5.3 Bug Fixes

- Handle congested DoQ streams ¶ References: #13631, pull request 13638
- Make sure we enforce the ACL over DoQ and DoH3 ¶ References: pull request 13670
- Set the DNS over HTTP/3 default port to 443 ¶ References: pull request 13647
- Grant unidirectional HTTP/3 streams for DoH3 ¶ References: pull request 13678
- Buffer HTTP/3 headers until the query has been dispatched ¶ References: #13687, pull request 13689
- Add content-type header information in DoH3 responses ¶ References: #13690, pull request 13713
- Properly set the incoming protocol when logging via Protobuf or dnstap ¶ References: pull request 13716
- Fix the ‘TCP Died Reading Query” metric, as reported by Coverity ¶ References: pull request 13630

21.6 1.8.3

Released: 15th of December 2023

Please review the *Upgrade Guide* before upgrading from versions < 1.8.x.

21.6.1 Improvements

- Add a *DynBlockRulesGroup:removeRange()* binding ¶ References: pull request 13601
- Add a *DNSHeader:getTC()* Lua binding ¶ References: pull request 13605

21.6.2 Bug Fixes

- Fix code producing JSON ¶ References: #13050, pull request 13607
- Refactor the exponential back-off timer code ¶ References: #13519, pull request 13523
- Detect and dismiss truncated UDP responses from a backend ¶ References: pull request 13598
- Fix the removal of the last rule by name or UUID ¶ References: pull request 13599

- Fix several cosmetic issues in eBPF dynamic blocks, update documentation ¶ References: #13307, pull request 13602

21.7 1.9.0-alpha4

Released: 14th of December 2023

Please review the *Upgrade Guide* before upgrading.

21.7.1 New Features

- Add support for incoming DNS over HTTP/3 ¶ References: pull request 13556
- Add a 'rings' endpoint to the REST API ¶ References: pull request 13489
- Add support for setting Extended DNS Error statuses ¶ References: pull request 13473
- Add a cache-miss ratio dynamic block rule ¶ References: pull request 13492
- Add *getAddressInfo()* for asynchronous DNS resolution ¶ References: pull request 13505
- Add *PayloadSizeRule* and *TCResponseAction* ¶ References: pull request 13564

21.7.2 Improvements

- Require Quiche >= 0.15.0 ¶ References: pull request 13437
- Add missing DoQ latency metrics ¶ References: pull request 13472
- Send a HTTP 400 response to HTTP/1.1 clients ¶ References: pull request 13594
- Remove legacy terms from the codebase (Kees Monshouwer) ¶ References: pull request 13023
- Wrap *DIR** objects in unique pointers to prevent memory leaks ¶ References: pull request 13191
- Add a DynBlockRulesGroup:removeRange() binding ¶ References: pull request 13342
- Fix a few Coverity warnings ¶ References: pull request 13435
- Fix Coverity CID 1523748: Performance inefficiencies in dolog.hh ¶ References: pull request 13445
- Add *pdns::visit_directory()*, wrapping opendir/readdir/closedir ¶ References: #13191, pull request 13485
- Improve *NetmaskGroupRule/SuffixMatchNodeRule*, deprecate *makeRule* ¶ References: pull request 13500
- Add *NetmaskGroup:addNMG()* to merge Netmask groups ¶ References: pull request 13503
- Add an option to set the SSL proxy protocol TLV ¶ References: pull request 13506
- Add Proxy Protocol v2 support to *TeeAction* ¶ References: pull request 13509
- Allow setting the action from *setSuffixMatchRule{,FFI}()*'s visitor ¶ References: pull request 13515
- Allow enabling incoming PROXY protocol on a per-bind basis ¶ References: pull request 13517
- Make the max size of entries in the packet cache configurable ¶ References: pull request 13537
- Spoof a raw response for ANY queries ¶ References: pull request 13560
- Add Lua FFI bindings: hashing arbitrary data and knowing if the query was received over IPv6 ¶ References: pull request 13565
- Add *QNameSuffixRule* ¶ References: pull request 13592

21.7.3 Bug Fixes

- Fix building with DoQ but without DoH or DoT ¶ References: pull request 13524
- Fix the case where nghttp2 is available but DoH is disabled ¶ References: pull request 13381
- Fix the removal of the last rule by name or UUID ¶ References: pull request 13488
- Refactor the exponential back-off timer code ¶ References: pull request 13520
- Detect and dismiss truncated UDP responses from a backend ¶ References: pull request 13536

21.8 1.9.0-alpha3

Released: 20th of October 2023

Please review the *Upgrade Guide* before upgrading.

21.8.1 New Features

- Add support for incoming DNS over QUIC ¶ References: pull request 13280
- Log Extended DNS Errors (EDE) to protobuf ¶ References: pull request 13185

21.8.2 Improvements

- Display the rule name, if any, in the web interface ¶ References: pull request 13335
- Add Lua binding to downstream address (Denis Machard) ¶ References: #13201, pull request 13275
- Set proper levels when logging messages ¶ References: pull request 13305
- Fix several cosmetic issues in eBPF dynamic blocks, update documentation ¶ References: pull request 13310

21.8.3 Bug Fixes

- Fix a typo in ‘Client timeouts’ (phonedph1) ¶ References: pull request 13302
- Netmask: Normalize subnet masks coming from a string ¶ References: pull request 13340
- Prevent DNS header alignment issues ¶ References: #13280, pull request 13372

21.8.4 misc

- Fix timeouts on incoming DoH connections with nghttp2 ¶ References: pull request 13298
- Enable back h2o support in our packages ¶ References: pull request 13274

21.9 1.9.0-alpha2

Released: Never

21.10 1.8.2

Released: 11th of October 2023

This release fixes the HTTP2 rapid reset attack for the packages we provide. If you are compiling DNSdist yourself or using the packages provided by your distribution, please check that the h2o library has been patched to mitigate this vulnerability.

Please review the *Upgrade Guide* before upgrading from versions < 1.8.x.

21.10.1 Bug Fixes

- Switch to our fork of h2o to mitigate the HTTP2 rapid reset attack ¶ References: pull request #13349

21.11 1.7.5

Released: 11th of October 2023

This release fixes the HTTP2 rapid reset attack for the packages we provide. If you are compiling DNSdist yourself or using the packages provided by your distribution, please check that the h2o library has been patched to mitigate this vulnerability.

Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

21.11.1 Bug Fixes

- Switch to our fork of h2o to mitigate the HTTP2 rapid reset attack ¶ References: pull request #13351

21.12 1.9.0-alpha1

Released: 18th of September 2023

Please review the *Upgrade Guide* before upgrading.

21.12.1 New Features

- Add Lua bindings to access selector and action ¶ References: #13007, pull request 13013
- Add an option to write *grepq*'s output to a file ¶ References: pull request 12689

21.12.2 Improvements

- Add support for incoming DoH via nghttp2 ¶ References: pull request 12678
- Add metrics for health-check failures ¶ References: pull request 13009
- Fix building our fuzzing targets from a dist tarball ¶ References: pull request 13145
- Add a DNSHeader:getTC() Lua binding ¶ References: pull request 13135
- Stop passing -u dnsdist -g dnsdist on systemd's ExecStart ¶ References: pull request 13088
- Use arc4random only for random values ¶ References: pull request 12931

21.12.3 Removals

- Change the default for building with net-snmp from *auto* to *no* ¶ References: pull request 13168

21.13 1.8.1

Released: 8th of September 2023

Please review the *Upgrade Guide* before upgrading from versions < 1.8.x.

21.13.1 New Features

- Allow declaring custom metrics at runtime ¶ References: pull request 13123

21.13.2 Improvements

- Stop using the now deprecated ERR_load_CRYPTO_strings() to detect OpenSSL ¶ References: pull request 13121
- Automatically load Lua FFI inspection functions ¶ References: pull request 13122
- Increment the “dyn blocked” counter for eBPF blocks as well ¶ References: pull request 13125
- Make DNSQType.TSIG available (Jacob Bunk) ¶ References: pull request 13133

21.13.3 Bug Fixes

- Fix a crash when X-Forwarded-For overrides the initial source IP ¶ References: pull request 12977
- Fix cache hit and miss metrics with DoH queries ¶ References: #12762, pull request 13131
- Fix a memory leak when processing TLS tickets w/ OpenSSL 3.x ¶ References: pull request 13130
- Fix a race when creating the first TLS connections ¶ References: pull request 13178
- Print the received, invalid health-check response ID ¶ References: pull request 12820
- Account for the health-check run time between two runs ¶ References: pull request 12821
- Properly set the size of the UDP health-check response ¶ References: pull request 12822
- Add the query ID to health-check log messages, fix nits ¶ References: pull request 12823
- Stop setting SO_REUSEADDR on outgoing UDP client sockets ¶ References: pull request 12824
- Properly handle short reads on backend upgrade discovery ¶ References: pull request 13116
- Undo an accidentally change of disableZeroScope to disableZeroScoping (Winfried Angele) ¶ References: pull request 13117
- Fix the group of the dnsdist.conf file when installed via RPM ¶ References: #13027, pull request 13118
- Work around Red Hat 8 messing up OpenSSL’s headers and refusing to fix it ¶ References: #12926, pull request 13119
- Fix a typo for libedit in the dnsdist features list ¶ References: pull request 13120
- Fix webserver config template for our docker container (Houtworm) ¶ References: pull request 13124
- YaHTTP: Prevent integer overflow on very large chunks ¶ References: pull request 13127
- Fix the console description of PoolAction and QPSPoolAction (phonedph1) ¶ References: pull request 13128

- Properly handle reconnection failure for backend UDP sockets ¶ References: #12711, pull request 13129
- SpoofAction: copy the QClass from the request (Christof Chen) ¶ References: pull request 13132
- Properly record self-answered UDP responses with recvmmsg ¶ References: pull request 13150

21.14 1.7.4

Released: 14th of April 2023

Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

21.14.1 New Features

- Add getPoolNames() function, returning a list of pool names (Christof Chen) ¶ References: #12074, pull request 12621

21.14.2 Bug Fixes

- Fix the health-check timeout computation for DoH backend ¶ References: pull request 12327
- Skip invalid OCSP files after issuing a warning ¶ References: #12341, pull request 12421
- Ignore unclean TLS session shutdown ¶ References: #12236, pull request 12237
- Properly encode json strings containing binary data ¶ References: #9349, pull request 12260
- Properly update rcode-related metrics on RCodeAction hits ¶ References: #11498, pull request 12484
- Fix building with boost < 1.56 ¶ References: #12177, pull request 12183
- lock.hh: include <stdexcept> ¶ References: #12453, pull request 12460
- dnsdist-protocols.hh: include <cstdint> (Sander Hoentjen) ¶ References: pull request 12569
- Fix the formatting of 'showServers' ¶ References: pull request 12535
- Properly record the incoming flags on a timeout ¶ References: #11905, pull request 12529
- Prevent an underflow of the TCP d_queued counter ¶ References: #12357, pull request 12365
- Properly handle single-SOA XFR responses ¶ References: #12099, pull request 12100
- Also reconnect on ENETUNREACH. (Asgeir Storesund Nilsen) ¶ References: #4155, pull request 11830
- Fix a bug in SetEDNSOptionAction ¶ References: #11728, pull request 11729
- Fix the number of concurrent queries on a backend TCP conn ¶ References: pull request 11718

21.15 1.8.0

Released: 30th of March 2023

Please review the *Upgrade Guide* before upgrading from versions < 1.8.x.

21.15.1 Bug Fixes

- Fix 'Unknown key' issue for actions and rules parameters ¶ References: pull request 12687
- Fix a dnsheader unaligned case ¶ References: pull request 12672
- secpoll: explicitly include necessary ctime header for time_t ¶ References: pull request 12654

21.16 1.8.0-rc3

Released: 16th of March 2023

Please review the *Upgrade Guide* before upgrading from versions < 1.8.x.

21.16.1 Improvements

- Report per-incoming transport latencies in the web interface ¶ References: pull request 12638
- Report the TCP latency for TCP-only Do53, DoT and DoH backends ¶ References: pull request 12648
- Count hits in the StatNode ¶ References: pull request 12626

21.16.2 Bug Fixes

- Use the correct source address when harvesting failed ¶ References: pull request 12641
- Fix a race when a cross-protocol query triggers an IO error ¶ References: pull request 12639

21.17 1.8.0-rc2

Released: 9th of March 2023

Please review the *Upgrade Guide* before upgrading from versions < 1.8.x.

21.17.1 Improvements

- Add Lua bindings for PB requestorID, deviceName and deviceID ¶ References: pull request 12615
- Clean up the fortify and LTO m4 by not directly editing flags ¶ References: pull request 12593
- YaHTTP: Better detection of whether C++11 features are available ¶ References: pull request 12589
- Skip signal-unsafe logging when we are about to exit, with TSAN ¶ References: pull request 12587

21.17.2 Bug Fixes

- Fix compilation with DoH disabled (Adam Majer) ¶ References: pull request 12588
- Only increment the 'servfail-responses' metric on backend responses (phonedph1) ¶ References: pull request 12592
- Fix the harvesting of destination addresses ¶ References: pull request 12586

21.18 1.8.0-rc1

Released: 23rd of February 2023

Please review the *Upgrade Guide* before upgrading from versions < 1.8.x.

21.18.1 New Features

- Allow randomly selecting a backend UDP socket and query ID ¶ References: pull request 11163
- Dynamic discovery and upgrade of backends ¶ References: pull request 11293
- Add support for password protected PKCS12 files for TLS configuration ¶ References: pull request 11027
- Add experimental support for TLS asynchronous engines ¶ References: pull request 10734
- Add an API endpoint to remove entries from caches ¶ References: #10468, #6154, pull request 12473
- Add support for user defined metrics ¶ References: pull request 11674
- Add the ability to change the qname and owner names in DNS packets ¶ References: pull request 12417
- Implement async processing of queries and responses ¶ References: pull request 12388
- Add the ability to cap the TTL of records after insertion into the cache ¶ References: pull request 12384
- Add SetReducedTTLResponseAction ¶ References: pull request 12400
- Add a Lua FFI interface for metrics ¶ References: pull request 12385
- Add a new chain of rules triggered after cache insertion ¶ References: pull request 12280
- Added XDP middleware for dropped/redirected queries logging (Mini Pierre) ¶ References: pull request 11020
- Implement a 'lazy' health-checking mode ¶ References: pull request 12065
- Add getPoolNames() function, returning a list of pool names (Christof Chen) ¶ References: #12073, pull request 12074
- Cleaner way of getting the IP/masks associated to a network interface ¶ References: pull request 12082
- Add Lua helpers to look into the content of DNS payloads ¶ References: pull request 12022
- Add more Lua bindings for network-related operations ¶ References: pull request 11994
- Add Lua binding for inspecting the in-memory ring buffers ¶ References: pull request 12008
- Add Lua bindings to look up domain and IP addresses from the cache ¶ References: pull request 12007
- Implement *SuffixMatchTree::getBestMatch()* to get the name that matched ¶ References: pull request 11698
- Use BPF_MAP_TYPE_LPM_TRIE for range matching (Y7n05h) ¶ References: pull request 11526
- Add getVerbose() function ¶ References: pull request 11637
- Add Lua bindings to access the DNS payload as a string ¶ References: pull request 11606
- Add setVerbose() to switch the verbose mode at runtime ¶ References: pull request 11567
- Add a 'getAddressAndPort()' method to DOHFrontend and TLSFrontend objects ¶ References: #11434, pull request 11547
- Add setTCPFastOpenKey() (Y7n05h) ¶ References: #9994, pull request 11497
- Add Lua FFI helpers for protocol and MAC address access, proxy protocol payload generation ¶ References: pull request 11173
- Add support to store mac address in query rings ¶ References: pull request 11184
- Add newThread() function ¶ References: pull request 11126
- Lua support to remove resource records from a response ¶ References: pull request 11098
- Add support to spoof a full self-generated response from lua ¶ References: pull request 11051
- Add a Lua FFI helper to generate proxy protocol payloads ¶ References: pull request 10949
- Add Lua bindings to get the list of network interfaces, addresses ¶ References: pull request 11017
- Add lua support to limit TTL values of responses ¶ References: pull request 11059

21.18.2 Improvements

- Merge the 'main' and 'client' DoH threads in single acceptor mode ¶ References: pull request 12386
- Speed up DoH handling by preventing allocations and copies ¶ References: pull request 12000
- OpenSSL 3.0: Offer TLS providers as an alternative to TLS engines ¶ References: pull request 12423
- Skip invalid OCSP files after issuing a warning ¶ References: #12341, pull request 12421
- Gracefully handle a failure to create a TLS server context ¶ References: pull request 12435
- Ignore unclean TLS session shutdown ¶ References: #12236, pull request 12237
- Skip DoT/DoH frontend when a tls configuration error occurs ¶ References: pull request 11675
- More useful default ports for DoT/DoH backends ¶ References: pull request 11415
- Libssl: Load only the ciphers and digests needed for TLS, not all of them ¶ References: pull request 11166
- Add support for metadata in protobuf messages ¶ References: pull request 12520
- Enable experimental kTLS support with OpenSSL on Linux ¶ References: pull request 12545
- Improve the scalability of MaxQPSIPRule() ¶ References: pull request 12537
- Reduce useless wake-ups from the event loop ¶ References: pull request 12276
- Faster cache-lookups for DNS over HTTPS queries ¶ References: pull request 11901
- Add a 'single acceptor thread' build option, reducing the number of threads ¶ References: pull request 12003
- Make recording queries/responses in the ringbuffers optional ¶ References: pull request 11883
- Slightly reduce contention around a pool's servers ¶ References: pull request 11852
- Only call getsockname() once per incoming DoH connection ¶ References: pull request 11851
- Set TCP_NODELAY on the TCP connection to backends ¶ References: pull request 11734
- Avoid allocating memory in LB policies for small number of servers ¶ References: pull request 11689
- SuffixMatchTree: Improve lookup performance ¶ References: pull request 11624
- Change dns_tolower() and dns_toupper() to use a table ¶ References: pull request 11655
- Scan the UDP buckets only when we have outstanding queries ¶ References: #11576, pull request 11577
- Prevent allocations in two corner cases ¶ References: pull request 11531
- Only allocate the health-check mplexer when needed ¶ References: #11422, pull request 11437
- Defer the actual allocation of the ring buffer entries ¶ References: pull request 11171
- Add an option for unauthenticated access to the dashboard ¶ References: #10360, pull request 12474
- Add an option for unauthenticated access to the API ¶ References: pull request 11514
- Add support for custom prometheus names in custom metrics ¶ References: pull request 12553
- Slightly reduce the number of allocations in API calls ¶ References: pull request 11987
- Add more detailed metrics ¶ References: pull request 11716
- Compute backend latency earlier, to avoid internal latency ¶ References: pull request 11707
- Add 'statistics' to the general API endpoint ¶ References: pull request 11659
- Add a counter for the number of cache cleanups ¶ References: pull request 11656
- Enable Link-Time Optimization for our packages ¶ References: pull request 12543
- Stop using the deprecated *boost::optional::get_value_or* ¶ References: pull request 12538
- List version number early ¶ References: #10932, pull request 12530

- Remove duplicate code in xdp (Y7n05h) ¶ References: pull request 12518
- Warn on unsupported parameters (Aki Tuomi) ¶ References: pull request 10115
- Add unit tests for the Lua FFI interface ¶ References: #12417, pull request 12469
- Refactor 'cannot be used at runtime' handling ¶ References: pull request 12492
- Fail if we can't check the configuration file ¶ References: #7611, pull request 12481
- Add a configure option to enable LTO ¶ References: pull request 12441
- Add a new configure option to initialize automatic variables ¶ References: pull request 12427
- Enable FORTIFY_SOURCE=3 when supported by the compiler ¶ References: pull request 12381
- Proper accounting of response and cache hits ¶ References: pull request 12405
- Support OpenSSL 3.0 for ipcipher CA6 encryption/decryption ¶ References: pull request 12411
- Stronger guarantees against data race in the UDP path ¶ References: pull request 12383
- Add bindings for the current and query times in DQ/DR ¶ References: pull request 12402
- Raise RLIMIT_MEMLOCK automatically when eBPF is requested (Yogesh Singh) ¶ References: pull request 11554
- Systemd: Add "After" dependency on time-sync.target (Kevin P. Fleming) ¶ References: #11153, pull request 12248
- DNSName constructor use memchr instead of strchr and cleanup with string_view (Axel Viala) ¶ References: pull request 11863
- Fix building with boost < 1.56 ¶ References: #12142, pull request 12177
- Retain output when expunging from multiple caches (Christof Chen) ¶ References: #12075, pull request 12077
- Add build-time options to disable the dynamic blocks and UDP response delay ¶ References: pull request 11993
- Add missing thread names ¶ References: pull request 11992
- Add a build option (define) to prevent loading OpenSSL's errors ¶ References: pull request 11988
- Properly load ciphers and digests with OpenSSL 3.0 ¶ References: #11853, pull request 11862
- Add local ComboAddress parameter for SBind() at TeeAction() (@FredericDT) ¶ References: pull request 11889
- Do not keep the mplexer created for the initial health-check around ¶ References: pull request 11844
- Use getrandom() if available ¶ References: pull request 11723
- Implement a limit of concurrent connections to a backend ¶ References: pull request 11713
- Fill ringbuffers with responses served from the cache ¶ References: #11585, pull request 11712
- Bind to the requested src interface without a src address ¶ References: pull request 11696
- Log listening addresses and version at the 'info' level ¶ References: pull request 11711
- Refactor sendfromto (Y7n05h) ¶ References: pull request 11651
- Optionally send 'verbose' messages to a file, and log them at 'DEBUG' level otherwise ¶ References: pull request 11668
- Log when exiting due to a SIGTERM signal ¶ References: pull request 11669
- Add the protocol (Do53, DoT, DoH, . . .) of backends in the API ¶ References: pull request 11673
- Remove implicit type conversion (Y7n05h) ¶ References: #11619, pull request 11620
- Log when a console message exceeds the maximum size ¶ References: #11488, pull request 11543

- Include the address of the backend in ‘relayed to’ messages ¶ References: pull request 11578
- Better log message when no downstream server are available ¶ References: pull request 11573
- Raise the number of entries in a packet cache to at least 1 ¶ References: #11383, pull request 11546
- Merge multiple parameters in newBPFFilter (Y7n05h) ¶ References: #11526, pull request 11535
- Reject BPFFilter::attachToAllBinds() at configuration time (Y7n05h) ¶ References: pull request 11523
- Add more build-time options to select features ¶ References: pull request 11515
- Multiplexer: Take the maximum number of events as a hint ¶ References: pull request 11517
- Add –log-timestamps flag ¶ References: pull request 11388
- Add a parameter to PoolAction to keep processing rules ¶ References: pull request 11174
- Fix build with OpenSSL 3.0.0 ¶ References: pull request 11196
- Build with *-fvisibility=hidden* by default ¶ References: pull request 11178
- Add a lot more of build-time options to select features ¶ References: pull request 10950

21.18.3 Bug Fixes

- Apply the max number of concurrent conns per client to DoH ¶ References: #12019, pull request 12483
- Fix the health-check timeout computation for DoH backend ¶ References: pull request 12327
- Fix a crash on a invalid protocol in DoH forwarded-for header ¶ References: #11604, pull request 11621
- Better handling of multiple carbon servers ¶ References: #10517, #11216, pull request 12424
- Include <cstdint> in dnsdist-protocols.hh (Sander Hoentjen) ¶ References: pull request 12569
- Fix the formatting of ‘showServers’ ¶ References: pull request 12535
- Properly record the incoming flags on a timeout ¶ References: #11905, pull request 12529
- Properly update rcode-related metrics on RCodeAction hits ¶ References: #11498, pull request 12484
- Handle out-of-memory exceptions in the UDP receiver thread ¶ References: pull request 12387
- Prevent an underflow of the TCP d_queued counter ¶ References: #12357, pull request 12365
- Properly handle single-SOA XFR responses ¶ References: #12099, pull request 12100
- Fix a bug in SetEDNSOptionAction ¶ References: #11728, pull request 11729
- Also reconnect on ENETUNREACH. (Asgeir Storesund Nilsen) ¶ References: #4155, pull request 11830
- Keep retained capabilities even when switching user/group ¶ References: pull request 11761
- Fix the number of concurrent queries on a backend TCP conn ¶ References: pull request 11718
- Fix invalid proxy protocol payload on a DoH TC to TCP retry ¶ References: pull request 11604
- Use the correct outgoing protocol in our ring buffers ¶ References: #11501, pull request 11545

21.18.4 Removals

- Remove the leak warning with GnuTLS >= 3.7.3 ¶ References: #11201, pull request 11324

21.19 1.7.3

Released: 2nd of November 2022

Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

dnsdist 1.7.3 contains no functional changes or bugfixes. This release strictly serves to bring dnsdist packages to our EL9 and Ubuntu Jammy repositories, and upgrades the dnsdist Docker image from Debian buster to Debian bullseye, as buster is officially EOL.

21.19.1 Improvements

- add el9/9stream targets ¶ References: pull request 11948
- docker images: upgrade to Debian bullseye ¶ References: pull request 11974
- dh_builddeb: force gzip compression (this makes the Ubuntu Jammy packages compatible with our Debian-hosted repositories) ¶ References: pull request 11742

21.20 1.7.2

Released: 14th of June 2022

Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

21.20.1 Improvements

- Scan the UDP buckets only when we have outstanding queries ¶ References: #11576, pull request 11579
- Only allocate the health-check mplexer when needed ¶ References: #11422, pull request 11580
- Add Lua bindings to access the DNS payload as a string ¶ References: #11606, pull request 11666

21.20.2 Bug Fixes

- Fix invalid proxy protocol payload on a DoH TC to TCP retry ¶ References: #11604, pull request 11665
- Fix a crash on a invalid protocol in DoH forwarded-for header ¶ References: #11621, pull request 11667
- Add missing descriptions for prometheus metrics ¶ References: #11602, pull request 11664

21.21 1.7.1

Released: 25th of April 2022

Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

21.21.1 Improvements

- Remove the leak warning with GnuTLS >= 3.7.3 ¶ References: #11201, pull request 11324
- Fix compilation with OpenSSL 3.0.0 ¶ References: pull request 11195
- Docker images: remove capability requirements ¶ References: #11081, pull request 11094
- Docker image: install ca-certificates ¶ References: #11290, pull request 11292

- Work around a compiler bug seen on OpenBSD/amd64 using clang-13 ¶ References: #11113, pull request 11176
- Stop using the now deprecated and useless std::binary_function ¶ References: pull request 11197
- Add a 'getAddressAndPort()' method to DOHFrontend and TLSFrontend objects ¶ References: #11434, pull request 11547

21.21.2 Bug Fixes

- Fix the health-check timeout for outgoing DoH connections ¶ References: #11250, pull request 11253
- Set Server Name Indication on outgoing TLS connections (DoT, DoH) ¶ References: #11249, pull request 11251
- Fix the latency-count metric ¶ References: #11239, pull request 11323
- Fix a use-after-free in case of a network error in the middle of a XFR query ¶ References: #11330, pull request 11335
- Properly use eBPF when the DynBlock is not set ¶ References: #11504, pull request 11550
- Fix 'inConfigCheck()' ¶ References: #11254, pull request 11255
- Use the correct outgoing protocol in our ring buffers ¶ References: #11501, pull request 11545
- Raise the number of entries in a packet cache to at least 1 ¶ References: #11383, pull request 11546
- Fix wrong eBPF values (qtype, counter) being inserted for qnames ¶ References: pull request 11565
- The check interval applies to health-check, not timeouts ¶ References: #11375, pull request 11572

21.22 1.7.0

Released: 17th of January 2022

Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

21.22.1 Bug Fixes

- Test the correct member in DynBlockRatioRule::warningRatioExceeded (Doug Freed) ¶ References: #11131, pull request 11156

21.23 1.7.0-rc1

Released: 22nd of December 2021

Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

21.23.1 Improvements

- Reuse and save the TLS session tickets in DoT healthchecks ¶ References: pull request 11037

21.23.2 Bug Fixes

- Fix a double-free when a DoH cross-protocol response is dropped ¶ References: pull request 11075
- Check the size of the query when re-sending a DoH query ¶ References: pull request 11079

21.24 1.7.0-beta2

Released: 29th of November 2021

21.24.1 Improvements

- Add a function to know how many TLS sessions are currently cached ¶ References: pull request 10997
- Warn that GnuTLS 3.7.x leaks memory when validating certs ¶ References: pull request 11001
- Add a function to set the UDP recv/snd buffer sizes ¶ References: #10898, pull request 11008
- Add 'showWebserverConfig' ¶ References: #10135, pull request 11006

21.24.2 Bug Fixes

- Fix a memory leak when reusing TLS tickets for outgoing connections ¶ References: pull request 10999
- Fix compiler/static analyzer warnings ¶ References: #10988, pull request 10993
- Fix Lua parameters bound checks ¶ References: pull request 11007
- Add missing visibility attribute on *dnsdist_ffi_dnsquestion_get_qname_hash* ¶ References: pull request 11031

21.25 1.7.0-beta1

Released: 16th of November 2021

Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

21.25.1 New Features

- Implement filesystem pinning for eBPF maps, drop and truncate via XDP (Pierre Grié) ¶ References: pull request 10498, pull request 10883
- Add range support for dynamic blocks ¶ References: #4993, pull request 10815
- Add the ability to retain select capabilities at runtime ¶ References: pull request 10923

21.25.2 Improvements

- Support DoT, DoH and DNSCrypt transports for protobuf and dnstap ¶ References: #9103, pull request 10879
- Use the same outgoing TCP connection for different clients ¶ References: pull request 10862
- Read as many DoH responses as possible before yielding ¶ References: pull request 10875
- Stop over-allocating for DoH queries ¶ References: pull request 10876
- Convert make_pair to emplace (Rosen Penev) ¶ References: pull request 10646
- Add syslog identifier to service file ¶ References: #10651, pull request 10795
- Get rid of make_pair (Rosen Penev) ¶ References: pull request 10868
- Use make_unique instead of new (Rosen Penev) ¶ References: pull request 10870
- Handle existing EDNS content for SetMacAddrAction/SetEDNSOptionAction ¶ References: #4670, pull request 10907

21.25.3 Bug Fixes

- Keep watching idle DoH backend connections ¶ References: pull request 10845
- Fix the cleaning of TCP, DoT and DoH connections to the backend ¶ References: pull request 10920
- Properly handle I/O exceptions in the health checker ¶ References: pull request 10874
- NetmaskTree: Drop the 'noexcept' qualifier on the TreeNode ctor ¶ References: pull request 10900
- Fix build without nghttp2 ¶ References: pull request 10922
- Remove debug print line flooding logs (Eugen Mayer) ¶ References: pull request 10935
- Credentials: EVP_PKEY_CTX_set1_scrypt_salt() takes an *unsigned char** ¶ References: #10938, pull request 10943

21.26 1.7.0-alpha2

Released: 19th of October 2021

Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

21.26.1 New Features

- Add lua support for SetEDNSOptionAction ¶ References: pull request 10814
- Rule for basing decisions on outstanding queries in a pool (phonedph1) ¶ References: pull request 10832

21.26.2 Improvements

- Disable TLS renegotiation, release buffers for outgoing TLS ¶ References: pull request 10823
- Don't create SSLKEYLOGFILE files with wide permissions ¶ References: pull request 10760
- Update existing tags when calling setTagAction and setTagResponseAction ¶ References: pull request 10767
- Fix the unit tests to handle v4-only or v6-only connectivity ¶ References: #10403, pull request 10775
- Improve the coverage of the outgoing DoH code ¶ References: pull request 10782
- Allow skipping arbitrary EDNS options when computing packet hash ¶ References: pull request 10791
- Add incoming and outgoing protocols to grepq ¶ References: pull request 10833
- Allow setting the block reason from the SMT callback ¶ References: #10559, pull request 10835
- Clear the UDP states of TCP-only backends ¶ References: pull request 10844
- Replace shared by unique ptrs, reduce structs size ¶ References: pull request 10846

21.26.3 Bug Fixes

- Better handling of outgoing DoH workers ¶ References: #10771, pull request 10772
- Properly cache UDP queries passed to a TCP/DoT/DoH backend ¶ References: pull request 10787
- Use per-thread credentials for GnuTLS client connections ¶ References: pull request 10841
- Only set recursion protection once we know we do not return ¶ References: pull request 10848

21.27 1.7.0-alpha1

Released: 23rd of September 2021

> Please review the *Upgrade Guide* before upgrading from versions < 1.7.x.

21.27.1 New Features

- Implementation of DoH between dnsdist and the backend ¶ References: pull request 10635
- Implement cross-protocol queries, including outgoing DNS over TLS ¶ References: pull request 10338
- Add support for Lua per-thread FFI rules and actions ¶ References: pull request 10501
- Add FFI functions to spoof multiple raw values ¶ References: #10456, pull request 10532
- Add support for range-based lookups into a Key-Value store ¶ References: #10520, pull request 10525
- Implement SpoofSVCAction to return SVC responses ¶ References: #10367, pull request 10597

21.27.2 Improvements

- Don't look up the LMDB dbi by name for every query ¶ References: pull request 10520
- Move to hashed passwords for the web interface ¶ References: #7937, pull request 10157
- Fix 'temporary used in loop' warnings reported by g++ 11.1.0 ¶ References: pull request 10429
- Skip some memory allocations in client mode to reduce memory usage ¶ References: pull request 10441
- Support multiple ip addresses for dnsdist-resolver lua script (Wim) ¶ References: pull request 10414
- Make DNSDist XFR aware when transfer is finished (Dimitrios Mavrommatis) ¶ References: #10436, pull request 10489
- Do not report latency metrics of down upstream servers (Holger Hoffstätte) ¶ References: #10500, pull request 10508
- Carry the exact incoming protocol (Do53, DNSCrypt, DoT, DoH) in DQ ¶ References: #10338, pull request 10537
- Implement 'reload()' to rotate Log(Response)Action's log file ¶ References: #10502, pull request 10527
- Document that setECSOverride has its drawbacks (Andreas Jakum) ¶ References: pull request 10626
- Convert dnsdist and the recursor to LockGuarded ¶ References: pull request 10649
- Handle waiting for a descriptor to become readable OR writable ¶ References: pull request 10631
- Clean up a bit of "cast from type [. . .] casts away qualifiers" warnings ¶ References: pull request 10687
- Reorganize the IDState and Rings fields to reduce memory usage

  ¶ References: pull request 10381

21.27.3 Bug Fixes

- Catch FDMultiplexerException in IOStateHandler's destructor ¶ References: pull request 10656
- Resizing LMDB map size while there might be open transactions is unsafe ¶ References: pull request 10672
- Ignore TCAction over TCP ¶ References: #10693, pull request 10695
- Stop raising the number of TCP workers to the number of TCP binds ¶ References: pull request 10704
- Handle exception raised in IOStateGuard's destructor ¶ References: pull request 10724

21.28 1.6.1

Released: 15th of September 2021

Please review the *Upgrade Guide* before upgrading from versions < 1.6.x.

21.28.1 New Features

- Add the missing DOHFronted::loadNewCertificatesAndKeys() ¶ References: #10418, pull request 10550
- Implement a web endpoint to get metrics for only one pool ¶ References: #10482, pull request 10560

21.28.2 Bug Fixes

- Set the dnstap/protobuf transport to TCP for DoH queries ¶ References: #10497, pull request 10538
- Backport a missing mutex header ¶ References: pull request 10438
- Properly handle ECS for queries with ancount or nscount > 0 ¶ References: #10419, pull request 10619
- Catch FDMultiplexerException in IOStateHandler's destructor ¶ References: pull request 10656
- Fix outstanding counter issue on TCP error ¶ References: #10705, pull request 10706

21.29 1.6.0

Released: 11th of May 2021

21.30 1.5.2

Released: 10th of May 2021

Please review the *Upgrade Guide* before upgrading from versions < 1.5.x.

21.30.1 Bug Fixes

- Fix a crash when a DoH responses map is updated at runtime ¶ References: #9934, pull request 9936
- Fix SNI on resumed sessions by acknowledging the name sent by the client ¶ References: #9921, pull request 9922
- Fix the DNSName move assignment operator ¶ References: pull request 9749
- Fix a typo in prometheus metrics dnsdist_frontend_tlshandshakefailures #9728 (AppliedPrivacy) ¶ References: #9728, pull request 9729
- Make: two fixes ¶ References: pull request 9583
- Fix eBPF filtering of long qnames ¶ References: #9689, pull request 9717
- Fix a hang when removing a server with more than one socket ¶ References: pull request 9900
- Fix Dynamic Block RCode rules messing up the queries count ¶ References: #9756, pull request 9980
- Fix EDNS in ServFail generated when no server is available ¶ References: #10006, pull request 10012
- Prevent a crash with DynBPF objects in client mode ¶ References: #10090, pull request 10095
- Add missing getEDNSOptions and getDO bindings for DNSResponse ¶ References: pull request 10355

21.31 1.6.0-rc2

Released: 4th of May 2021

Please review the *Upgrade Guide* before upgrading from versions < 1.6.x.

21.31.1 Improvements

- Make the backend queryLoad and dropRate values atomic ¶ References: pull request 10323

21.31.2 Bug Fixes

- Fix missing locks in DNSCrypt certificates management ¶ References: pull request 10346
- Only use eBPF for "drop" actions, clean up more often ¶ References: #10324, pull request 10327

21.32 1.6.0-rc1

Released: 20th of April 2021

Please review the *Upgrade Guide* before upgrading from versions < 1.6.x.

21.32.1 Improvements

- Replace pthread_rwlock with std::shared_mutex ¶ References: #10209, pull request 10216
- Also disable PMTU for v6 ¶ References: pull request 10264

21.32.2 Bug Fixes

- Lua: don't destroy keys during table iteration ¶ References: pull request 10171
- Add missing getEDNSOptions and getDO bindings for DNSResponse ¶ References: #10262, pull request 10267
- Fix some issues reported by Thread Sanitizer ¶ References: pull request 10274

21.33 1.6.0-alpha3

Released: 29th of March 2021

Please review the *Upgrade Guide* before upgrading from versions < 1.6.x.

21.33.1 Improvements

- Set OpenSSL to release buffers when idle, saves 35 kB per connection ¶ References: pull request 10179
- Unify certificate reloading syntaxes ¶ References: pull request 10214
- Disable TLS renegotiation by default ¶ References: pull request 10218
- Improve TCP connection reuse, add metrics ¶ References: pull request 10156
- Using DATA to report memory usage is unreliable, start using RES instead, as it seems reliable and relevant ¶ References: #7591, pull request 10161

- Add a metric for TCP listen queue full events ¶ References: pull request 10184
- Enable sharding by default, greater pipe buffer sizes ¶ References: pull request 10204
- Add limits for cached TCP connections, metrics ¶ References: pull request 10207

21.33.2 Bug Fixes

- Fix the handling of DoH queries with a non-zero ID ¶ References: pull request 10208
- Fix the TCP connect timeout, add metrics ¶ References: pull request 10201

21.34 1.6.0-alpha2

Released: 4th of March 2021

> Please review the *Upgrade Guide* before upgrading from versions < 1.6.x.

21.34.1 New Features

- Add option to spoofRawAction to spoof multiple answers (Sander Hoentjen) ¶ References: pull request 10063
- Add 'spoof' and 'spoofRaw' Lua bindings ¶ References: pull request 10073

21.34.2 Improvements

- Make NetmaskTree::fork() a bit easier to understand ¶ References: #10035, pull request 10046
- Do not update the TCP error counters on idle states ¶ References: pull request 10131
- > Bind __tostring instead of toString for Lua, so that conversion to string works automatically (Aki Tuomi)

  ¶ References: pull request 9361

21.34.3 Bug Fixes

- Remove forgotten debug line in the web server ¶ References: #10049, pull request 10050
- Create TCP worker threads before acceptors ones ¶ References: pull request 10088
- Prevent a crash with DynBPF objects in client mode ¶ References: #10090, pull request 10095
- Fix several bugs in the TCP code path, add unit tests ¶ References: pull request 10108
- Fix size check during trailing data addition, regression tests ¶ References: pull request 10139
- Clean up expired entries from all the packet cache's shards ¶ References: pull request 10133

21.35 1.6.0-alpha1

Released: 2nd of February 2021

Please review the *Upgrade Guide* before upgrading from versions < 1.6.x.

21.35.1 New Features

- Add per-thread Lua FFI load-balancing policies ¶ References: pull request 9175
- Implement Lua custom web endpoints ¶ References: #9120, pull request 9676
- Implement TCP out-of-order ¶ References: pull request 9582
- Add support for incoming Proxy Protocol ¶ References: pull request 9616
- Add SkipCacheResponseAction ¶ References: #9536, pull request 9960

21.35.2 Improvements

- Use more of systemd's sandboxing options when available ¶ References: pull request 8969
- Add an option to allow sub-paths for DoH ¶ References: pull request 9962
- Prioritize ChaCha20-Poly1305 when client does (Sukhbir Singh) ¶ References: pull request 9510
- Start all TCP worker threads on startup ¶ References: pull request 9957
- Speed up the round robin policy ¶ References: pull request 9382
- Avoid unnecessary allocations and copies with DNSName::toDNSString() ¶ References: pull request 9424
- Get rid of allocations in the packet cache's fast path ¶ References: #8993, pull request 9420
- Fix the DNSName move assignment operator ¶ References: pull request 9749
- Don't copy the policy for every query ¶ References: pull request 9850
- UUID: Use the non-cryptographic variant of the boost::uuid ¶ References: pull request 9832
- Use an eBPF filter for Dynamic blocks when available ¶ References: #6763, #9756, pull request 9782
- Use protozero for Protocol Buffer operations ¶ References: #9780, #9781, pull request 9843
- Limit the number of concurrent console and web connections ¶ References: #4978, pull request 9997
- Add prometheus metrics for top Dynamic Blocks entries ¶ References: pull request 9756
- Add per connection queries count and duration stats for DoH ¶ References: pull request 9738
- Add Lua bindings to get a server's latency ¶ References: pull request 9273
- Wrap more FILE objects in smart pointers ¶ References: pull request 9225
- Set the default EDNS buffer size on generated answers to 1232 ¶ References: pull request 9049
- Add support for FreeBSD's SO_REUSEPORT_LB ¶ References: #9156, pull request 9157
- Accept string in DNSDistPacketCache:expungeByName ¶ References: pull request 9428
- DNSName: add toDNSString convenience function ¶ References: pull request 9466
- Skip EDNS Cookies in the packet cache ¶ References: #5131, pull request 8993
- Add the query payload size to the verbose log over TCP ¶ References: pull request 9677
- Add the response code in the packet cache dump ¶ References: #9274, pull request 9737
- Add an optional name to rules ¶ References: pull request 9746
- Add the ability to set ACL from a file (Matti Hiljanen) ¶ References: pull request 9822
- Add a Lua binding for the number of queries dropped by a server ¶ References: #9861, pull request 9862
- Move to c++17 ¶ References: pull request 9913
- Fix warnings on autoconf 2.70 ¶ References: #9918, pull request 9920
- Reduce diff to upstream yahttp, fixing a few CodeQL reports ¶ References: pull request 9955

- Handle syslog facility as string, document the numerical one ¶ References: #9383, pull request 9989
- Deprecate parameters to webserver(), add 'statsRequireAuthentication' parameter ¶ References: #8710, #9311, pull request 9972
- Add a counter for queries truncated because of a rule ¶ References: #9357, pull request 9992
- Replace offensive terms in our code and documentation ¶ References: pull request 9993
- Use aligned atomics to prevent false sharing ¶ References: #9455, pull request 9998
- Unify non-terminal actions as SetXXXAction() ¶ References: #8118, pull request 9974
- Accept a NMG to fill DynBlockRulesGroup ranges ¶ References: #9545, pull request 10015
- Silence clang 12 warning ¶ References: pull request 10023
- Fix a few warnings reported by clang's static analyzer and cppcheck ¶ References: pull request 10035

21.35.3 Bug Fixes

- Fix a crash when a DoH responses map is updated at runtime ¶ References: #9927, pull request 9934
- Fix SNI on resumed sessions by acknowledging the name sent by the client ¶ References: pull request 9921
- Use toStringWithPort instead of manual addr/port concat (Mischan Toosarani-Hausberger) ¶ References: #9075, pull request 9222
- Force a reconnection when a downstream transitions to the UP state (Nuitari, Stephane Bakhos) ¶ References: pull request 9275
- Handle EINTR in DelayPipe ¶ References: pull request 9381
- Handle empty DNSNames in grepq() ¶ References: pull request 9431
- Make: two fixes ¶ References: pull request 9583
- Fix eBPF filtering of long qnames ¶ References: #9626, pull request 9689
- Improve const-correctness of Lua bindings (Georgeto) ¶ References: pull request 9721
- Fix a hang when removing a server with more than one socket ¶ References: pull request 9900
- Appease clang++ 12 ASAN on macOS ¶ References: pull request 9925
- Bunch of signed vs unsigned warnings ¶ References: pull request 9937
- Send a NotImp answer on empty (qdcount=0) queries ¶ References: #9961, pull request 9991
- Don't apply QPS to backend server on cache hits ¶ References: #7038, pull request 9999
- Fix EDNS in ServFail generated when no server is available ¶ References: #10006, pull request 10012

21.35.4 Removals

- Rename topRule() and friends ¶ References: pull request 9532
- Remove useless second argument for *SpoofAction* ¶ References: #9783, pull request 9784

21.36 1.5.1

Released: 1st of October 2020

Please review the *Upgrade Guide* before upgrading from versions < 1.5.x.

21.36.1 Improvements

- Add the 'clearConsoleHistory' command ¶ References: #9372, pull request 9540

21.36.2 Bug Fixes

- Stop the related responder thread when a backend is removed ¶ References: #9372, pull request 9541
- Fix getEDNSOptions() for {AN,NS}COUNT != 0 and ARCOUNT = 0 ¶ References: pull request 9542
- Fix building with LLVM11 (@RvdE) ¶ References: pull request 9543
- Only add EDNS on negative answers if the query had EDNS ¶ References: pull request 9555

21.37 1.5.0

Released: 30th of July 2020

Please review the *Upgrade Guide* before upgrading from versions < 1.5.x.

21.37.1 Improvements

- Use explicit flag for the specific version of c++ we are targeting. ¶ References: pull request 9231
- Prevent a copy of a pool's backends when selecting a server. ¶ References: pull request 9360

21.37.2 Bug Fixes

- Fix compilation with h2o_socket_get_ssl_server_name(). ¶ References: pull request 9344
- Prevent a possible overflow via large Proxy Protocol values. (Valentei Sergey) ¶ References: pull request 9320
- Avoid name clashes on Solaris derived systems. ¶ References: #9279, pull request 9348
- Resize hostname to final size in getCarbonHostname(). (Aki Tuomi) ¶ References: pull request 9343
- Fix compilation on OpenBSD/amd64. ¶ References: pull request 9346
- Handle calling PacketCache methods on a nil object. ¶ References: pull request 9356

21.38 1.5.0-rc4

Released: 7th of July 2020

Please review the *Upgrade Guide* before upgrading from versions < 1.5.x.

21.38.1 Bug Fixes

- Prevent a race between the DoH handling threads ¶ References: pull request 9278

21.39 1.5.0-rc3

Released: 18th of June 2020

Please review the *Upgrade Guide* before upgrading from versions < 1.5.x.

21.39.1 New Features

- Implement an ACL in the internal web server ¶ References: pull request 9229

21.39.2 Improvements

- Less negatives in secpoll error messages improves readability. ¶ References: pull request 9100
- Use std::string_view when available (Rosen Penev) ¶ References: pull request 9207
- Clean up dnsdistconf.lua as a default configuration file ¶ References: #8038, pull request 9238
- Add optional masks to KeyValueLookupKeySourceIP ¶ References: pull request 9244

21.39.3 Bug Fixes

- Use non-blocking pipes to pass DoH queries/responses around ¶ References: #9206, pull request 9211
- Fix compilation on systems that do not define HOST_NAME_MAX ¶ References: #9125, pull request 9127
- Do not use *using namespace std;* ¶ References: pull request 9213

21.40 1.5.0-rc2

Released: 13th of May 2020

Please review the *Upgrade Guide* before upgrading from versions < 1.5.x.

21.40.1 Improvements

- Add the unit to the help for latency buckets ¶ References: pull request 9084
- Avoid copies in for loops ¶ References: pull request 9042
- Build with -Wmissing-declarations -Wredundant-decls ¶ References: pull request 9054
- Use std::shuffle instead of std::random_shuffle ¶ References: #9004, pull request 9016
- Get rid of a naked pointer in the /dev/poll event multiplexer ¶ References: pull request 9053
- A few warnings fixed, reported by clang on OpenBSD ¶ References: pull request 9059
- Wrap pthread objects ¶ References: pull request 9067
- NetmaskTree: do not test node for null, the loop guarantees node is not null. ¶ References: pull request 9078

21.40.2 Bug Fixes

- Fix duplicated HTTP/1 counter in 'showDOHFrontends()' ¶ References: pull request 9068
- Fix compilation of the ports event multiplexer ¶ References: #9025, pull request 9031
- Gracefully handle a failure to remove FD on (re)-connection ¶ References: pull request 9057

21.41 1.5.0-rc1

Released: 16th of April 2020

Please review the *Upgrade Guide* before upgrading from versions < 1.5.x.

21.41.1 Improvements

- Expose SuffixMatchNode::remove in Lua ¶ References: pull request 8956
- Remove a std::move() preventing Return-Value Optimization in lmdb-safe.cc ¶ References: pull request 8962
- Drop responses with the QR bit set to 0 ¶ References: pull request 8996
- Add an option to control the size of the TCP listen queue ¶ References: #8986, pull request 8994

21.41.2 Bug Fixes

- Keep accepting fragmented UDP datagrams on DNSCrypt binds ¶ References: pull request 8974
- Accept UDP datagrams larger than 1500 bytes for DNSCrypt ¶ References: #8974, pull request 8976
- On OpenBSD string_view is both in boost and std ¶ References: pull request 8955

21.42 1.5.0-alpha1

Released: 20th of March 2020

Please review the *Upgrade Guide* before upgrading from versions < 1.5.x.

21.42.1 New Features

- Implement LuaFFIRule, LuaFFIAction and LuaFFIResponseAction ¶ References: #7617, pull request 8505
- Add SetNegativeAndSOAAction() and its Lua binding ¶ References: #4747, pull request 8171
- Implement dynamic blocking on ratio of rcode/total responses ¶ References: pull request 8274
- Add bounded loads to the consistent hashing policy ¶ References: #7387, pull request 8567
- LogResponseAction (phonedph1) ¶ References: pull request 8654
- Add spoofRawAction() to craft answers from raw bytes ¶ References: pull request 8722
- Add support for Proxy Protocol between dnsdist and the recursor ¶ References: pull request 8874
- Implement bounded loads for the whashed and wrandom policies ¶ References: pull request 8909

21.42.2 Improvements

- Don’t accept sub-paths of configured DoH URLs ¶ References: #8573, pull request 8760
- Implement Cache-Control headers in DoH ¶ References: #8586, pull request 8762
- Change the default DoH path from / to /dns-query ¶ References: #8819, pull request 8905
- Add support for the processing of X-Forwarded-For headers ¶ References: #8661, pull request 8945
- Switch the default DoT provider from GnuTLS to OpenSSL ¶ References: pull request 8380
- Document that the ‘keyLogFile’ option requires OpenSSL >= 1.1.1 ¶ References: #8806, pull request 8899

- Add the source and destination ports to the protobuf msg ¶ References: pull request 8702
- Better handling of reconnections in Remote Logger ¶ References: pull request 8887
- Rework NetmaskTree for better CPU and memory efficiency. (Stephan Bosch) ¶ References: pull request 8355
- Implement parallel health checks ¶ References: pull request 8491
- Use move semantics when updating the content of the StateHolder ¶ References: pull request 8538
- Keep a masked network in the Netmask class ¶ References: pull request 8812
- Make FrameStream IO parameters configurable ¶ References: pull request 8937
- Add backend status to prometheus metrics ¶ References: #8746, pull request 8772
- Add 'IO wait' and 'steal' metrics on Linux ¶ References: pull request 8783
- Don't start as root within a systemd environment ¶ References: pull request 7820
- Separate the check-config and client modes ¶ References: pull request 8456
- Add the number of received bytes to StatNode entries ¶ References: pull request 8529
- Support setting the value of AA, AD and RA when self-generating answers ¶ References: #8534, pull request 8556
- pthread_rwlock_init() should be matched by pthread_rwlock_destroy() ¶ References: pull request 8580
- Replace include guard ifdef/define with pragma once (Chris Hofstaedtler) ¶ References: pull request 8631
- Allow retrieving and deleting a backend via its UUID ¶ References: pull request 8657
- Load an openssl configuration file, if any, during startup ¶ References: pull request 8733
- Add get*BindCount() functions ¶ References: pull request 8848
- Add sessionTimeout setting for TLS session lifetime (Matti Hiljanen) ¶ References: pull request 8882
- Detect {Libre,Open}SSL functions availability during configure ¶ References: #8739, pull request 8900
- Warn on startup about low weight values with chashed ¶ References: #8669, pull request 8950

21.42.3 Bug Fixes

- Set the DoH ticket rotation delay before loading tickets ¶ References: pull request 8949
- Display the correct DoT provider ¶ References: pull request 8662
- Use ref counting for the DoT TLS context ¶ References: pull request 8761
- Add 'queue full' metrics for our remote logger, log at debug only ¶ References: #8629, pull request 8883
- Fix ECS addition when the OPT record is not the last one ¶ References: #8098, pull request 8115
- Wait longer for the TLS ticket to arrive in our tests ¶ References: pull request 8591
- Add missing exception message in KVS error ¶ References: pull request 8604
- Add getTag()/setTag() Lua bindings for a DNSResponse ¶ References: pull request 8782
- Fix key logging for DNS over TLS ¶ References: #8442, pull request 8787
- Fix a typo in the help/completion for getDNSCryptBindCount ¶ References: pull request 8855
- Implement rmACL() (swoga) ¶ References: pull request 8856
- Remove unused lambda capture reported by clang++ ¶ References: pull request 8879

21.43 1.4.0

Released: 20th of November 2019

Please review the *Upgrade Guide* before upgrading from versions < 1.4.x.

21.43.1 Improvements

- Fix the default value of `setMaxUDPOutstanding` in the console's help (phonedph1) ¶ References: pull request 8531
- Add bindings for the noerrors and drops members of StatNode ¶ References: pull request 8522
- Fix -Wshadow warnings (Aki Tuomi) ¶ References: pull request 8440
- Fix typo: settting to setting (Chris Hofstaedtler) ¶ References: pull request 8509

21.43.2 Bug Fixes

- Lowercase the name blocked by a SMT dynamic block ¶ References: pull request 8524

21.43.3 misc

- Prefer the cipher suite from the server by default (DoH, DoT) ¶ References: pull request 8526

21.44 1.4.0-rc5

Released: 30th of October 2019

Please review the *Upgrade Guide* before upgrading from versions < 1.4.x.

21.44.1 Improvements

- Rename the 'address' label to 'frontend' for DoH metrics ¶ References: pull request 8465

21.44.2 Bug Fixes

- Increment the DOHUnit ref count when it's set in the IDState ¶ References: pull request 8471

21.45 1.4.0-rc4

Released: 25th of October 2019

Please review the *Upgrade Guide* before upgrading from versions < 1.4.x.

21.45.1 New Features

- Add support dumping TLS keys via keyLogFile ¶ References: pull request 8442

21.45.2 Improvements

- Implement reference counting for the DOHUnit object ¶ References: pull request 8416
- Lowercase custom DoH header names ¶ References: #8353, pull request 8365
- Add metrics about TLS handshake failures for DoH and DoT ¶ References: pull request 8447
- Merge the setup of TLS contexts in DoH and DoT ¶ References: pull request 8383
- Add metrics about unknown/inactive TLS ticket keys ¶ References: pull request 8406
- Add metrics about TLS versions with DNS over TLS ¶ References: pull request 8387
- Add a 'preferServerCiphers' option for DoH and DoT ¶ References: pull request 8382
- Count the number of concurrent connections for DoH as well ¶ References: pull request 8395
- Refactor DoH prometheus metrics again ¶ References: pull request 8361
- Add more options to LogAction (non-verbose mode, timestamps) ¶ References: #8390, pull request 8411
- Fix formatting in showTCPStats() ¶ References: pull request 8415
- Use SO_BINDTODEVICE when available for newServer's source interface ¶ References: pull request 8372
- Check the address supplied to 'webserver' in check-config ¶ References: #8362, pull request 8364

21.45.3 Bug Fixes

- Clear the DoH session ticket encryption key in the ctor ¶ References: pull request 8388
- Add missing prometheus descriptions for cache-related metrics ¶ References: pull request 8409
- Add a prometheus 'thread' label to distinguish identical frontends ¶ References: pull request 8381
- Fix a typo in the prometheus description of 'senderrors' ¶ References: pull request 8378
- More prometheus fixes ¶ References: pull request 8368
- Fix the caching of large entries ¶ References: pull request 8408
- Work around cmsg_space somehow not being a constexpr on macOS ¶ References: #8412, pull request 8413
- Fix the creation order of rules when inserted via setRules() ¶ References: pull request 8359

21.46 1.4.0-rc3

Released: 30th of September 2019

Please review the *Upgrade Guide* before upgrading from versions < 1.4.x.

21.46.1 Improvements

- Allow accepting DoH queries over HTTP instead of HTTPS ¶ References: pull request 8267
- Implement TLS session ticket keys management for DoH ¶ References: pull request 8349
- Display the DoH and DoT binds in the web view ¶ References: pull request 8264
- Clean up our interactions with errno ¶ References: #7845, pull request 8083
- Remove the 'blockfilter' stat from the web view ¶ References: #5514, pull request 8265
- Fix some spelling mistakes noticed by lintian (Chris Hofstaedtler) ¶ References: pull request 8268

- dnsdistconf.lua use non-deprecated versions for 1.4.0 (phonedph1) ¶ References: pull request 8285
- Better use of labels in our DoH prometheus export ¶ References: pull request 8318

21.46.2 Bug Fixes

- Fix the newCDBKVStore console completion when LMDB is not enabled (phonedph1) ¶ References: pull request 8281
- Allow configure CDB_CFLAGS to work (phonedph1) ¶ References: pull request 8283
- Fix the warning message on an invalid secpoll answer ¶ References: pull request 8303
- Don't connect to remote logger in client/command mode ¶ References: #8300, pull request 8304

21.47 1.4.0-rc2

Released: 2nd of September 2019

Please review the *Upgrade Guide* before upgrading from versions < 1.4.x.

21.47.1 New Features

- Add support for early DoH HTTP responses ¶ References: pull request 8206
- Add a KeyValueStoreLookup action based on CDB or LMDB ¶ References: pull request 8139

21.47.2 Improvements

- Add minTLSVersion for DoH and DoT ¶ References: #8202, pull request 8207
- Split dnsdist-lua-bindings.cc to reduce memory consumption during compilation ¶ References: pull request 8250
- Add a Lua binding for *dynBlockRulesGroup:setQuiet(quiet)* ¶ References: pull request 8252

21.47.3 misc

- Update h2o to 2.2.6, fixing CVE-2019-9512, CVE-2019-9514 and CVE-2019-9515 for repo.powerdns.com packages ¶ References: pull request 8200

21.48 1.4.0-rc1

Released: 12th of August 2019

Please review the *Upgrade Guide* before upgrading from versions < 1.4.x.

21.48.1 New Features

- Add support for custom DoH headers (Melissa Voegeli) ¶ References: #7900, #7957, pull request 8148
- Add lua bindings, rules and action for DoH ¶ References: #8133, pull request 8153
- Add OCSP stapling (from files) for DoT and DoH ¶ References: #7812, pull request 8141
- Implement ContinueAction() ¶ References: pull request 8117

21.48.2 Improvements

- Send better HTTP status codes, handle ACL drops earlier ¶ References: pull request 7917
- Add more stats about DoH HTTP responses ¶ References: #7898, pull request 7933
- Improve error messages for DoT issues ¶ References: pull request 7978
- Accept more than one certificate in *addDNSCryptBind()* ¶ References: #8020, pull request 8042
- Disallow TCP disablement ¶ References: pull request 7860
- Update boost.m4 to the latest version ¶ References: pull request 7862
- Print stats from expungeByName (Matti Hiljanen) ¶ References: pull request 7909
- Squelch unused function warning ¶ References: #7950, pull request 7952
- SuffixMatchNode:add(): accept more types ¶ References: pull request 7985
- Explicitly align the buffer used for cmsgs ¶ References: #7981, pull request 7990
- Add *quiet* parameter to NetmaskGroupRule ¶ References: pull request 7992
- Clear cmsg_space(sizeof(data)) in cmsghdr to appease Valgrind ¶ References: #7981, pull request 7996
- Add static assertions for the size of the src address control buffer ¶ References: pull request 8007
- Don't create temporary strings to escape DNSName labels ¶ References: pull request 8013
- Display TCP/DoT queries and responses in verbose mode, opcode in grepq ¶ References: pull request 8024
- Be a bit more explicit about what failed in testCrypto() ¶ References: pull request 8025
- Update URLs to use HTTPS scheme (Chris Hofstaedtler) ¶ References: pull request 8110
- Double-check we only increment the outstanding counter once ¶ References: pull request 8113
- ext/ipcrypt: ship license in tarballs (Chris Hofstaedtler) ¶ References: #8108, pull request 8135
- Use a counter to mark IDState usage instead of the FD ¶ References: pull request 8154
- Increase the default value of setMaxUDPOutstanding to 65535 ¶ References: pull request 8175

21.48.3 Bug Fixes

- Properly override the HTTP Server header for DoH ¶ References: #7894, pull request 7911
- Proper HTTP response for timeouts over DoH ¶ References: #7917, pull request 7927
- Prevent a dangling DOHUnit pointer when send() failed ¶ References: pull request 8112
- Exit when requested DoT/DoH support is not compiled in ¶ References: pull request 7915
- Skip non-dnscrypt binds in *showDNSCryptBinds()* ¶ References: #8014, pull request 8015
- SuffixMatchTree: fix root removal, partial match of non-leaf nodes ¶ References: pull request 7886
- Deduplicate frontends entries with carbon and prometheus ¶ References: #7933, pull request 7934
- Update boost.m4 ¶ References: #6942, #8084, pull request 7951
- Fix short IOs over TCP ¶ References: #7971, pull request 7974
- Fix handling of backend connection failing over TCP ¶ References: pull request 7979
- Insert the response into the ringbuffer right after sending it ¶ References: pull request 8003
- Handle ENOTCONN on read() over TCP ¶ References: #8021, pull request 8030
- Make sure we always compile with BOOST_CB_ENABLE_DEBUG set to 0 ¶ References: pull request 8067
- Catch exceptions thrown when handling a TCP response ¶ References: pull request 8078

- Fix unlimited retries when TCP Fast Open is enabled ¶ References: pull request 8079
- M4/systemd.m4: fail when systemctl is not available ¶ References: pull request 8081
- Fix a typo in the Server's latency description for Prometheus (phonedph1) ¶ References: pull request 8105
- Console: flush cout after printing g_outputbuffer (Doug Freed) ¶ References: #8130, pull request 8131
- Fix signedness issue in isEDNSOptionInOpt() ¶ References: pull request 8158

21.49 1.4.0-beta1

Released: 6th of June 2019

Please review the *Upgrade Guide* before upgrading from versions < 1.4.x.

21.49.1 New Features

- Implement SNIRule for DoT and DoH ¶ References: #7210, pull request 7825

21.49.2 Improvements

- Support Prometheus latency histograms (Marlin Cremers) ¶ References: #6088, pull request 7853

21.49.3 Bug Fixes

- DoH: Don't let 'self' dangling while parsing the request's qname, this could lead to a crash ¶ References: #7810, pull request 7814
- Fix minor issues reported by Coverity ¶ References: pull request 7823
- Remove second, incomplete copy of lua EDNSOptionCode table ¶ References: pull request 7833

21.50 1.4.0-alpha2

Released: 26th of April 2019

Please review the *Upgrade Guide* before upgrading from versions < 1.4.x.

21.50.1 New Features

- Add DNS over HTTPS support based on libh2o ¶ References: #6911, #7526, pull request 7726

21.50.2 Improvements

- Ignore Path MTU discovery on UDP server socket ¶ References: pull request 7410
- Alternative solution to the unaligned accesses. ¶ References: pull request 7708

21.50.3 Bug Fixes

- Exit when setting ciphers fails (GnuTLS) ¶ References: pull request 7718

21.51 1.4.0-alpha1

Released: 12th of April 2019

> Please review the *Upgrade Guide* before upgrading from versions < 1.4.x.

21.51.1 New Features

- Make recursor & dnsdist communicate (ECS) 'variable' status ¶ References: pull request 7209
- Add namespace and instance variable to carbon key (Gibheer) ¶ References: #2362, #6941, pull request 6959
- Allow NoRecurse for use in dynamic blocks or Lua rules (phonedph1) ¶ References: pull request 7087
- Expose secpoll status ¶ References: #7194, pull request 7197
- Add an optional 'checkTimeout' parameter to 'newServer()' ¶ References: #7236, pull request 7323
- Add a 'rise' parameter to 'newServer()' ¶ References: #7237, pull request 7322
- Add a 'keepStaleData' option to the packet cache ¶ References: #7239, pull request 7310
- Expose trailing data (Richard Gibson) ¶ References: #6846, #6897, pull request 6967
- Add option to set interval between health checks (1848) ¶ References: pull request 7142
- Add EDNS unknown version handling (Dmitry Alenichev) ¶ References: pull request 7406
- DNSNameSet and QNameSetRule (Andrey) ¶ References: pull request 7537
- Add support for encrypting ip addresses #gdpr ¶ References: #6242, pull request 7481
- Add 'setSyslogFacility()' ¶ References: #5653, pull request 7677
- Add 'reloadAllCertificates()' ¶ References: pull request 7676

21.51.2 Improvements

- Fix warnings, mostly unused parameters, reported by -wextra ¶ References: pull request 7168
- Add optional uuid column to showServers() ¶ References: pull request 7191
- Configure –enable-pdns-option –with-third-party-module (Josh Soref) ¶ References: pull request 7026
- Drop remaining capabilities after startup ¶ References: pull request 7138
- More sandboxing using systemd's features ¶ References: pull request 6634
- Reduce systemcall usage in Protobuf logging ¶ References: pull request 7428
- Resync YaHTTP code to cmouse/yahttp@11be77a1fc4032 (Chris Hofstaedtler) ¶ References: pull request 7433
- Pass empty response (Dmitry Alenichev) ¶ References: pull request 7431
- Change the way getRealMemusage() works on linux (using statm) ¶ References: pull request 7502
- Prevent 0-ttl cache hits ¶ References: #7534, pull request 7585
- Add addDynBlockSMT() support to dynBlockRulesGroup ¶ References: #7139, pull request 7343
- Add frontend response statistics (Matti Hiljanen) ¶ References: pull request 7578
- Remove addLuaAction and addLuaResponseAction ¶ References: pull request 7670
- Refactoring of the TCP stack ¶ References: #4814, #7526, pull request 7559
- Prevent a conflict with BADSIG being clobbered ¶ References: #7556, pull request 7692

- Switch to the new 'newPacketCache()' syntax for 1.4.0 ¶ References: pull request 7689
- Move constants to proper namespace ¶ References: pull request 7678
- Unify the management of DNS/DNSCrypt/DoT frontends ¶ References: pull request 7694
- Fix compiler warning about returning garbage (Adam Majer)

  ¶ References: pull request 7167

21.51.3 Bug Fixes

- Protect GnuTLS tickets key rotation with a read-write lock ¶ References: pull request 7256
- Check that `SO_ATTACH_BPF` is defined before enabling eBPF ¶ References: pull request 7267
- Fix off-by-one in mvRule counting ¶ References: pull request 7426
- Don't convert nsec to usec if we need nsec ¶ References: pull request 7520
- Fix setRules() ¶ References: pull request 7594
- Handle EAGAIN in the GnuTLS DNS over TLS provider ¶ References: pull request 7560
- Gracefully handle a null latency in the webserver's js ¶ References: #7461, pull request 7586
- EDNSOptionView improvements ¶ References: pull request 7652
- Honor libcrypto include path ¶ References: #7481, pull request 7674

21.52 1.3.3

Released: 8th of November 2018

Please review the *Upgrade Guide* before upgrading from versions < 1.3.x.

21.52.1 New Features

- Add consistent hash builtin policy ¶ References: #6932, pull request 6737, pull request 6939
- Add EDNSOptionRule ¶ References: pull request 6803
- Add DSTPortRule (phonedph1) ¶ References: pull request 6813
- Make getOutstanding usable from both lua and console (phonedph1) ¶ References: pull request 6826
- Added :excludeRange and :includeRange methods to DynBPFFilter class (Reinier Schoof) ¶ References: pull request 6856
- Add Prometheus stats support (Pavel Odintsov, Kai S) ¶ References: #4947, #6002, pull request 3935, pull request 6343, pull request 6901, pull request 7007, pull request 7089
- Name threads in the programs ¶ References: #6974, pull request 6997
- Support the NXDomain action with dynamic blocks ¶ References: #6908, pull request 7075
- Add security polling ¶ References: pull request 7115
- Add a PoolAvailableRule to easily add backup pools (Robin Geuze) ¶ References: pull request 7140

21.52.2 Improvements

- Get rid of some allocs/copies in DNS parsing ¶ References: pull request 6831
- Set a correct EDNS OPT RR for self-generated answers ¶ References: #4857, #6348, pull request 6847
- Fix a sign-comparison warning in isEDNSOptionInOPT() ¶ References: pull request 6877
- Add warning rates to DynBlockRulesGroup rules ¶ References: #6907, pull request 6986
- Add support for exporting a server id in protobuf ¶ References: #6990, #7004, pull request 7015
- dnsdist did not set TCP_NODELAY, causing needless latency ¶ References: pull request 7030
- Add a setting to control the number of stored sessions ¶ References: pull request 7062
- Wrap GnuTLS and OpenSSL pointers in smart pointers ¶ References: #7060, pull request 7064
- Add a 'creationOrder' field to rules ¶ References: #6909, pull request 7078
- Fix return-type detection with boost 1.69's tribool ¶ References: #7091, pull request 7092
- Fix format string issue on 32bits ARM ¶ References: #7096, pull request 7104
- Wrap TCP connection objects in smart pointers ¶ References: pull request 7108
- Add the setConsoleOutputMaxMsgSize function ¶ References: #7084, pull request 7109
- Add the ability to update webserver credentials ¶ References: #7112, pull request 7117

21.52.3 Bug Fixes

- Display dynblocks' default action, None, as the global one ¶ References: pull request 6835
- Fix compilation when SO_REUSEPORT is not defined ¶ References: pull request 6956
- Release memory on DNS over TLS handshake failure ¶ References: pull request 7060
- Handle trailing data correctly when adding OPT or ECS info ¶ References: #6896, pull request 7165

21.53 1.3.2

Released: 10th of July 2018

Please review the *Upgrade Guide* before upgrading from versions < 1.3.x.

21.53.1 Bug Fixes

- Add missing include for PRId64, fix build on CentOS 6 / SLES 12 ¶ References: pull request 6785

21.54 1.3.1

Released: 10th of July 2018

Please review the *Upgrade Guide* before upgrading from versions < 1.3.x.

21.54.1 New Features

- Add support for more than one TLS certificate ¶ References: #6450, pull request 6524
- Add a negative ttl option to the packet cache ¶ References: #6579, pull request 6740
- Add the ability to dump a summary of the cache content ¶ References: pull request 6749
- Add netmask-based {ex,in}clusions to DynblockRulesGroup ¶ References: pull request 6760
- Add DNSAction.NoOp to debug dynamic blocks ¶ References: #6703, pull request 6776
- Add SetECSAction to set an arbitrary outgoing ecs value ¶ References: #6404, pull request 6734
- Add support for rotating certificates and keys ¶ References: pull request 6764

21.54.2 Improvements

- Remove *thelog* and *thel* and replace this with a global g_log ¶ References: #6357, pull request 6358
- Fix two small nits on the documentation ¶ References: pull request 6422
- Move the el6 dnsdist package to upstart ¶ References: #6394, pull request 6426
- CLI option improvements (Chris Hofstaedtler) ¶ References: #6433, pull request 6435
- Split pdns_enable_unit_tests (Chris Hofstaedtler) ¶ References: pull request 6436
- Re-do lua detection ¶ References: #6423, pull request 6445, pull request 6457, pull request 6470
- Docs: fix missing ref in the dnsdist docs ¶ References: pull request 6460
- Be more permissive in wrandom tests, log values on failure ¶ References: pull request 6502
- Tests: avoid failure on not-so-optimal distribution ¶ References: #6430, pull request 6523
- Add syntax to dns.proto to silence compilation warning. ¶ References: pull request 6577
- Fix warnings reported by gcc 8.1.0 ¶ References: pull request 6590
- Document setVerboseHealthchecks() ¶ References: #6483, pull request 6592
- Update dq.rst (phonedph1) ¶ References: pull request 6615
- Fix rpm scriptlets ¶ References: pull request 6641
- Don’t copy unitialized values of SuffixMatchTree ¶ References: pull request 6637
- Expose toString of various objects to Lua (Chris Hofstaedtler) ¶ References: pull request 6684
- Remove ‘expired’ states from MaxQPSIPRule ¶ References: pull request 6674
- Mark the remote member of DownstreamState as const ¶ References: #6664, pull request 6688
- Test the content of dynamic blocks using the API ¶ References: #6706, pull request 6710
- Default set “connection: close” header for web requests ¶ References: #6532, pull request 6711
- Update timedipsetrule.rst (phonedph1) ¶ References: pull request 6717
- Don’t access the TCP buffer vector past its size ¶ References: #6712, pull request 6716
- Show droprate in API output ¶ References: pull request 6563
- Refuse console connection without a proper key set ¶ References: #6683, #6709, pull request 6715
- Use LRU to clean the MaxQPSIPRule’s store ¶ References: pull request 6726
- Disable maybe uninitialized warnings with boost optional ¶ References: pull request 6769
- Luawrapper: report caught std::exception as lua_error ¶ References: #6541, pull request 6658
- Dnstap.rst: fix some editing errors (Chris Hofstaedtler) ¶ References: pull request 6602

- Allow known exception types to be converted to string ¶ References: #6535, pull request 6541

21.54.3 Bug Fixes

- Initialize the done variable in the rings' unit tests ¶ References: pull request 6425
- Reorder headers to fix OpenBSD build ¶ References: pull request 6429
- Restrict value range for weight parameter, avoid sum overflows dropping queries (Dan McCombs) ¶ References: pull request 6448
- Fix reconnection handling ¶ References: pull request 6672
- Dynamic blocks were being created with the wrong duration (David Freedman) ¶ References: pull request 6706
- Limit qps and latency to two decimals in the web view ¶ References: #6442, pull request 6718
- Check the flags to detect collisions in the packet cache ¶ References: pull request 6747
- Fix iterating over the results of exceed*() functions ¶ References: pull request 6762
- Fix duration false positive in the dynblock regression tests ¶ References: pull request 6767
- Implement NoneAction() ¶ References: #6758, pull request 6775
- Detect ECS collisions in the packet cache ¶ References: #6747, pull request 6754
- Fix an outstanding counter race when reusing states ¶ References: pull request 6773

21.55 1.3.0

Released: 30th of March 2018

Please review the *Upgrade Guide* before upgrading from versions < 1.3.x.

21.55.1 New Features

- Add an optional *status* parameter to `Server:setAuto()`. ¶ References: pull request 5625
- Add `inClientStartup()` function. ¶ References: pull request 6072
- Add tag-based routing of queries. ¶ References: pull request 6037
- Add experimental *DNS-over-TLS* support. ¶ References: pull request 6117, pull request 6175, pull request 6176, pull request 6177, pull request 6189
- Add simple *dnstap* support (Justin Valentini, Chris Hofstaedtler). ¶ References: pull request 5201, pull request 6170
- Add experimental XPF support based on draft-bellis-dnsop-xpf-04. ¶ References: #5079, #5654, pull request 5594, pull request 6220
- Add `ERCodeRule()` to match on extended RCodes (Chris Hofstaedtler). ¶ References: pull request 6147
- Add `TempFailureCacheTTLAction()` (Chris Hofstaedtler). ¶ References: pull request 6003
- Add *DynBlockRulesGroup* to improve processing speed of the `maintenance()` function by reducing memory usage and not walking the ringbuffers multiple times. ¶ References: pull request 6391
- Add `console ACL` functions. ¶ References: #4654, pull request 6399
- Allow adding `EDNS Client Subnet information` to a query before looking in the cache. This allows serving ECS enabled answers from the cache when all servers in a pool are down. ¶ References: #6098, pull request 6400

21.55.2 Improvements

- Add cache sharding, `recvmmsg` and CPU pinning support. With these, the scalability of **dnsdist** is drastically improved. ¶ References: #5202, #5859, pull request 5576, pull request 5860
- Add burst option to *`MaxQPSIPRule()`* (42wim). ¶ References: pull request 5970
- Add Pools, cacheHitResponseRules to the API. ¶ References: pull request 6022
- Add a class option to health checks. ¶ References: #5748, pull request 5929
- Add UUIDs to rules, this allows tracking rules through modifications and moving them around. ¶ References: pull request 6030
- Apply ResponseRules to locally generated answers (Chris Hofstaedtler). ¶ References: #6182, pull request 6185
- Report *`LuaAction()`* and *`LuaResponseAction()`* failures in the log and send SERVFAIL instead of not answering the query (Chris Hofstaedtler). ¶ References: pull request 6283
- Unify global statistics accounting (Chris Hofstaedtler). ¶ References: pull request 6289
- Speed up the processing of large ring buffers. This change will make **dnsdist** more scalable with a large number of different clients. ¶ References: pull request 6350, pull request 6366
- Make custom `addLuaAction()` and `addLuaResponseAction()` callback's second return value optional. ¶ References: #6346, pull request 6363
- Add "server-up" metric count to Carbon Reporting (Lowell Mower). ¶ References: pull request 6327
- Add xchacha20 support for *DNSCrypt*. ¶ References: pull request 6045, pull request 6382
- Scalability improvement: Add an option to use several source ports towards a backend. ¶ References: pull request 6317
- Add '?' and 'help' for providing help() output on `dnsdist -c` (Kirill Ponomarev, Chris Hofstaedtler). ¶ References: #4845, pull request 5866, pull request 6375
- Replace the Lua mutex with a rw lock to limit contention. This improves the processing speed and parallelism of the policies. ¶ References: pull request 6190, pull request 6381
- Ensure **dnsdist** compiles on NetBSD (Tom Ivar Helbekkmo). ¶ References: pull request 6146
- Also log eBPF dynamic blocks, as regular dynamic block already are. ¶ References: #5845, pull request 5845
- Ensure large numbers are shown correctly in the API. ¶ References: #6211, pull request 6401
- Add option to *`showRules()`* to truncate the output length. ¶ References: #5763, pull request 6402
- Fix several warnings reported by clang's analyzer and cppcheck, should lead to small performance increases. ¶ References: pull request 6407

21.55.3 Bug Fixes

- Handle SNMP alarms so we can reconnect to the daemon. ¶ References: #5327, pull request 5328
- Fix signed/unsigned comparison warnings on ARM. ¶ References: #5489, pull request 5597
- Keep trying if the first connection to the remote logger failed ¶ References: pull request 5770
- Fix escaping unusual DNS label octets in DNSName is off by one (Kees Monshouwer). ¶ References: pull request 6018
- Avoid assertion errors in `NewServer()` (Chris Hofstaedtler). ¶ References: pull request 6403

21.55.4 Removals

- Remove the `--daemon` option from **dnsdist**. ¶ References: #6329, pull request 6394

21.56 1.2.1

Released: 16th of February 2018

Please review the *Upgrade Guide* before upgrading from versions < 1.2.x.

21.56.1 New Features

- Add configuration option to disable IP_BIND_ADDRESS_NO_PORT (Dan McCombs). ¶ References: pull request 5880

21.56.2 Improvements

- Handle bracketed IPv6 addresses without ports (Chris Hofstaedtler). ¶ References: pull request 6057

21.56.3 Bug Fixes

- Make dnsdist dynamic truncate do right thing on TCP/IP. ¶ References: pull request 5647
- Add missing QPSAction ¶ References: pull request 5686
- Don’t create a Remote Logger in client mode. ¶ References: pull request 5847
- Use libsodium’s CFLAGS, we might need them to find the includes. ¶ References: pull request 5858
- Keep the TCP connection open on cache hit, generated answers. ¶ References: pull request 6012
- Add the missing <sys/time.h> include to mplexer.hh for struct timeval. ¶ References: pull request 6041
- Sort the servers based on their ‘order’ after it has been set. ¶ References: pull request 6043
- Quiet unused variable warning on macOS (Chris Hofstaedtler). ¶ References: pull request 6073
- Fix the outstanding counter when an exception is raised. ¶ References: #5652, pull request 6094
- Do not connect the snmpAgent from a dnsdist client. ¶ References: #6163, pull request 6164

21.57 1.2.0

Released: 21st of August 2017

Please review the *Upgrade Guide* before upgrading from versions < 1.2.x.

21.57.1 New Features

- Add an option to export CNAME records over protobuf. ¶ References: #4709, pull request 4776
- Add TCP management options from **RFC 7766 section 10**. ¶ References: pull request 4611
- Add an option to ‘mute’ UDP responses per bind. ¶ References: #4527, pull request 4536
- Save history to home-dir, only use CWD as a last resort. ¶ References: #4562, pull request 4779
- Add the *`setRingBuffersSize()`* directive to allows changing the ringbuffer size. ¶ References: pull request 4898

- Allow TTL alteration via Lua. ¶ References: #4707, pull request 4787
- Add `RDRule()` to match queries with the `RD` flag set. ¶ References: pull request 4837
- Add `setWHashedPertubation()` for consistent `whashed` results. ¶ References: pull request 4897
- Add `tcpConnectTimeout` to `newServer()`. ¶ References: pull request 4818
- Add cache hit response rules. ¶ References: #4708, pull request 4788, pull request 5036
- Add *SNMP support*. ¶ References: pull request 4989, pull request 5123, pull request 5204
- Allow passing `DNSName`s as DNSRules. ¶ References: pull request 5070
- Add support for setting the server selection policy on a per pool basis (Robin Geuze). ¶ References: pull request 5113
- Add a `suffixMatch` parameter to `PacketCache:expungeByName()` (Robin Geuze). ¶ References: pull request 5159
- Add an option so the packet cache entries don't age. ¶ References: #5126, pull request 5136
- Add `QNameRule()`. ¶ References: pull request 5235
- Add an optional action to `addDynBlocks()`. ¶ References: pull request 5337
- Add an optional interface parameter to `addLocal()`/`setLocal()`. ¶ References: pull request 5344
- Make a `truncate` action available to DynBlock and Lua. ¶ References: pull request 5386
- Implement a runtime changeable rule that matches IP address for a certain time called `TimedIPSetRule()`. ¶ References: pull request 5336
- Add support for returning several IPs to spoof from Lua. ¶ References: pull request 5496
- Add Lua bindings to be able to rotate DNSCrypt keys, see *DNSCrypt*. ¶ References: #5420, #5507, pull request 5490, pull request 5508
- Add the capability to set arbitrary tags in protobuf messages. ¶ References: pull request 5396, pull request 5577
- Add setConsoleConnectionsLogging(). ¶ References: #5565, pull request 5581

21.57.2 Improvements

- Merge the client and server nonces to prevent replay attacks. ¶ References: pull request 4815
- Store the computed shared key and reuse it for the response for DNSCrypt messages. ¶ References: pull request 4813, pull request 4926
- Add `setTCPUseSinglePipe()` to use a single TCP waiting queue. ¶ References: pull request 4817
- Add `sendSizeAndMsgWithTimeout` to send size and data in a single call and use it for TCP Fast Open towards backends. ¶ References: #5494, pull request 4985, pull request 5501
- Tune systemd unit-file for medium-sized installations (Winfried Angele). ¶ References: pull request 4958
- Add the possibility to fill a `NetmaskGroup` (using `NetmaskGroup:addMask()`) from *exceeds** results. ¶ References: pull request 5185
- Add labels count to StatNode, only set the name once. ¶ References: pull request 5353
- DNSName: Check that both first two bits are set in compressed labels. ¶ References: #4851, pull request 4852
- Handle unreachable servers at startup, reconnect stale sockets ¶ References: #4131, #4155, pull request 4285
- Gracefully handle invalid addresses in `newServer()`. ¶ References: #4471, pull request 4474
- Use `IP_BIND_ADDRESS_NO_PORT` when available. ¶ References: pull request 4786

- Add an optional `seconds` parameter to `statNodeRespRing()`. ¶ References: #4660, #4775, pull request 4780
- Report a more specific lua version and report luajit in `--version`. ¶ References: pull request 4910
- Prevent issues by unshadowing variables. ¶ References: pull request 5056
- Register DNSName::chopOff (@plzz). ¶ References: pull request 4920
- Make `includeDirectory()` work sorted (Robin Geuze). ¶ References: #5053, pull request 5150, pull request 5171
- Allow embedded NULs in strings received from Lua. ¶ References: pull request 5147
- Cleanup closed TCP downstream connections. ¶ References: pull request 5163
- Improve reporting of C++ exceptions that bubble up via Lua. ¶ References: pull request 5230
- Add better logging on queries that get dropped, timed out or received. ¶ References: pull request 5253
- Print useful messages when query and response actions are mixed. ¶ References: pull request 5342
- Add `DNSRule::toString()` and add virtual destructors to DNSRule, DNSAction and DNSResponseAction so the destructors of derived classes are run even when deleted via the base type. ¶ References: pull request 5497
- Don't use square brackets for IPv6 in Carbon metrics. ¶ References: #5538, pull request 5579

21.57.3 Bug Fixes

- Unified `-k` and `setKey()` behaviour for client and server mode now. ¶ References: pull request 5199
- Refactor SuffixMatchNode using a SuffixMatchTree. ¶ References: #4761, pull request 4950
- Get rid of `std::move()` calls preventing copy elision. ¶ References: pull request 5359
- Send an HTTP 404 on unknown API paths. ¶ References: pull request 5089
- LuaWrapper: Use the correct index when storing a function. ¶ References: pull request 4775
- Send a latency of 0 over carbon, null over API for down servers. ¶ References: #4689, pull request 4785
- Fix negative port detection for IPv6 addresses on 32-bit. ¶ References: pull request 4911
- Fix crashed on SmartOS/Illumos (Roman Dayneko). ¶ References: #4579, pull request 4877
- Change `truncateTC` to defaulting to off, having it enabled by default causes an compatibility with **RFC 6891** (Robin Geuze). ¶ References: #4857, pull request 4859
- Don't cache answers without any TTL (like SERVFAIL). ¶ References: #4983, pull request 4987, pull request 5037
- Fix destination port reporting on "any" binds. ¶ References: pull request 5194
- Correctly truncate EDNS Client Subnetmasks. ¶ References: pull request 5320
- Fix `RecordsTypeCountRule()`'s handling of the # of records in a section. ¶ References: #5365, pull request 5369
- Change stats functions to always return lowercase names (Robin Geuze). ¶ References: #5287, pull request 5383
- Only use TCP Fast Open when supported and prevent compiler warnings. ¶ References: pull request 5449, pull request 5454
- Skip timeouts on the response latency graph. ¶ References: #5559, pull request 5563
- Copy the DNS header before encrypting it in place. ¶ References: #5566, pull request 5580

21.57.4 Removals

- Remove BlockFilter. ¶ References: #5513, pull request 5514
- Deprecate syntactic sugar functions. ¶ References: #5069, pull request 5526

21.57.5 misc

- Fix potential pointer wrap-around on 32 bits. ¶ References: pull request 5630
- Make the API available with an API key only. ¶ References: pull request 5631

21.58 1.1.0

Released December 29th 2016

Changes since 1.1.0-beta2:

21.58.1 Improvements

- #4783: Add -latomic on powerpc
- #4812: Handle header-only responses, handle Refused as Servfail in the cache

21.58.2 Bug fixes

- #4762: SuffixMatchNode: Fix an insertion issue for an existing node
- #4772: Fix dnsdist initscript config check

21.59 1.1.0-beta2

Released December 14th 2016

Changes since 1.1.0-beta1:

21.59.1 New features

- #4518: Fix dynblocks over TCP, allow refusing dyn blocked queries
- #4519: Allow altering the ECS behavior via rules and Lua
- #4535: Add `DNSQuestion:getDO()`
- #4653: `getStatisticsCounters()` to access counters from Lua
- #4657: Add `includeDirectory(dir)`
- #4658: Allow editing the ACL via the API
- #4702: Add `setUDPTimeout(n)`
- #4726: Add an option to return ServFail when no server is available
- #4748: Add `setCacheCleaningPercentage()`

21.59.2 Improvements

- #4533: Fix building with clang on OS X and FreeBSD
- #4537: Replace luawrapper's std::forward/std::make_tuple combo with std::forward_as_tuple (Sangwhan "fish" Moon)
- #4596: Change the default max number of queued TCP conns to 1000
- #4632: Improve dnsdist error message on a common typo/config mistake
- #4694: Don't use a const_iterator for erasing (fix compilation with some versions of gcc)
- #4715: Specify that dnsmessage.proto uses protobuf version 2
- #4765: Some service improvements

21.59.3 Bug fixes

- #4425: Fix a protobuf regression (requestor/responder mix-up) caused by a94673e
- #4541: Fix insertion issues in SuffixMatchTree, move it to dnsname.hh
- #4553: Flush output in single command client mode
- #4578: Fix destination address reporting
- #4640: Don't exit dnsdist on an exception in maintenance
- #4721: Handle exceptions in the UDP responder thread
- #4734: Add the TCP socket to the map only if the connection succeeds. Closes #4733
- #4742: Decrement the queued TCP conn count if writing to the pipe fails
- #4743: Ignore newBPFFilter() and newDynBPFFilter() in client mode
- #4753: Fix FD leak on TCP connection failure, handle TCP worker creation failure
- #4764: Prevent race while creating new TCP worker threads

21.60 1.1.0-beta1

Released September 1st 2016

Changes since 1.0.0:

21.60.1 New features

- #3762 Teeaction: send copy of query to second nameserver, sponge responses
- #3876 Add `showResponseRules()`, `{mv,rm,top}ResponseRule()`
- #3936 Filter on opcode, records count/type, trailing data
- #3975 Make dnsdist {A,I}XFR aware, document possible issues
- #4006 Add eBPF source address and qname/qtype filtering
- #4008 Node infrastructure for querying recent traffic
- #4042 Add server-side TCP Fast Open support
- #4050 Add `clearRules()` and `setRules()`
- #4114 Add `QNameLabelsCountRule()` and `QNameWireLengthRule()`
- #4116 Added src boolean to NetmaskGroupRule to match destination address (Reinier Schoof)

- #4175 Implemented query counting (Reinier Schoof)
- #4244 Add a `setCD` parameter to set cd=1 on health check queries
- #4284 Add RCodeRule(), Allow, Delay and Drop response actions
- #4305 Add an optional Lua callback for altering a Protobuf message
- #4309 Add showTCPStats function (RobinGeuze)
- #4329 Add options to LogAction() so it can append (instead of truncate) (Duane Wessels)

21.60.2 Improvements

- #3714 Add documentation links to dnsdist.service (Ruben Kerkhof)
- #3754 Allow the use of custom headers in the web server
- #3826 Implement a 'quiet' mode for SuffixMatchNodeRule()
- #3836 Log the content of webserver's exceptions
- #3858 Only log YaHTTP's parser exceptions in verbose mode
- #3877 Increase max FDs in systemd unit, warn if clearly too low
- #4019 Add an optional `addECS` option to `TeeAction()`
- #4029 Add version and feature information to version output
- #4079 Return an error on RemoteLog{,Response}Action() w/o protobuf
- #4246 API now sends pools as a JSON array instead of a string
- #4302 Add `help()` and `showVersion()`
- #4286 Add response rules to the API and Web status page
- #4068 Display the dyn eBPF filters stats in the web interface

21.60.3 Bug fixes

- #3755 Fix RegexRule example in dnsdistconf.lua
- #3773 Stop copying the HTTP request headers to the response
- #3837 Remove dnsdist service file on trusty
- #3840 Catch WrongTypeException in client mode
- #3906 Keep the servers ordered inside pools
- #3988 Fix `grepq()` output in the README
- #3992 Fix some typos in the AXFR/IXFR documentation
- #3995 Fix comparison between signed and unsigned integer
- #4049 Fix dnsdist rpm building script #4048 (Daniel Stirnimann)
- #4065 Include editline/readline.h instead of readline.h/history.h
- #4067 Disable eBPF support when BPF_FUNC_tail_call is not found
- #4069 Fix a buffer overflow when displaying an OpcodeRule
- #4101 Fix $ expansion in build-dnsdist-rpm
- #4198 newServer setting maxCheckFailures makes no sense (stutiredboy)
- #4205 Prevent the use of "any" addresses for downstream server

- #4220 Don't log an error when parsing an invalid UDP query
- #4348 Fix invalid outstanding count for {A,I}XFR over TCP
- #4365 Reset origFD asap to keep the outstanding count correct
- #4375 Tuple requires make_tuple to initialize
- #4380 Fix compilation with clang when eBPF support is enabled

21.61 1.0.0

Released April 21st 2016

Changes since 1.0.0-beta1:

21.61.1 Improvements

- #3700 Create user from the RPM package to drop privs
- #3712 Make check should run testrunner
- #3713 Remove contrib/dnsdist.service (Ruben Kerkhof)
- #3722 Use LT_INIT and disable static objects (Ruben Kerkhof)
- #3724 Include PDNS_CHECK_OS in configure (Chris Hofstaedtler)
- #3728 Document libedit Ctrl-R workaround for CentOS 6
- #3730 Make `topBandwidth()` behave like other top* functions
- #3731 Clarify a bit the documentation of load-balancing policies

21.61.2 Bug fixes

- #3711 Building rpm needs systemd headers (Ruben Kerkhof)
- #3736 Add missing Lua binding for NetmaskGroupRule()
- #3739 Drop privileges after daemonizing and writing our pid

21.62 1.0.0-beta1

Released April 14th 2016

Changes since 1.0.0-alpha2:

21.62.1 New features

- Per-pool packet cache
- Some actions do not stop the processing anymore when they match, allowing more complex setups: Delay, Disable Validation, Log, MacAddr, No Recurse and of course None
- The new RE2Rule() is available, using the RE2 regular expression library to match queries, in addition to the existing POSIX-based RegexRule()
- SpoofAction() now supports multiple A and AAAA records
- Remote logging of questions and answers via Protocol Buffer

21.62.2 Improvements

- #3405 Add health check logging, `maxCheckFailures` to backend
- #3412 Check config
- #3440 Client operation improvements
- #3466 Add dq binding for skipping packet cache in LuaAction (Jan Broer)
- #3499 Add support for multiple carbon servers
- #3504 Allow accessing the API with an optional API key
- #3556 Add an option to limit the number of queued TCP connections
- #3578 Add a `disable-syslog` option
- #3608 Export cache stats to carbon
- #3622 Display the ACL content on startup
- #3627 Remove ECS option from response's OPT RR when necessary
- #3633 Count "TTL too short" cache events
- #3677 systemd-notify support

21.62.3 Bug fixes

- #3388 Lock the Lua context before executing a LuaAction
- #3433 Check that the answer matches the initial query
- #3461 Fix crash when calling rmServer() with an invalid index
- #3550,#3551 Fix build failure on FreeBSD (Ruben Kerkhof)
- #3594 Prevent EOF error for empty console response w/o sodium
- #3634 Prevent dangling TCP fd in case setupTCPDownstream() fails
- #3641 Under threshold, QPS action should return None, not Allow
- #3658 Fix a race condition in MaxQPSIPRule

21.63 1.0.0-alpha2

Released February 5th 2016

Changes since 1.0.0-alpha1:

21.63.1 New features

- Lua functions now receive a DNSQuestion `dq` object instead of several parameters. This adds a greater compatibility with PowerDNS and allows adding more parameters without breaking the API (#3198)
- Added a `source` option to `newServer()` to specify the local address or interface used to contact a downstream server (#3138)
- CNAME and IPv6-only support have been added to spoofed responses (#3064)
- `grepq()` can be used to search for slow queries, along with `topSlow()`

- New Lua functions: `addDomainCNAMESpoof()`, `AllowAction()` by @bearggg, `exceedQRate()`,`MacAddrAction()`,`makeRule()`,`NotRule()`,`OrRule()`,`QClassRule()`, `RCodeAction()`, `SpoofCNAMEAction()`, `SuffixMatchNodeRule()`, `TCPRule()`, `topSlow()`
- `NetmaskGroup` support have been added in Lua (#3144)
- Added `MacAddrAction()` to add the source MAC address to the forwarded query (#3313)

21.63.2 Bug fixes

- An issue in DelayPipe could make dnsdist crash at startup
- `downstream-timeouts` metric was not always updated
- `truncateTC` was unproperly updating the response length (#3126)
- DNSCrypt responses larger than queries were unproperly truncated
- An issue prevented info message from being displayed in non-verbose mode, fixed by Jan Broer
- Reinstating an expired Dynamic Rule was not correctly logged (#3323)
- Initialized counters in the TCP client thread might have cause FD and memory leak, reported by Martin Pels (#3300)
- We now drop queries containing no question (qdcount == 0) (#3290)
- Outstanding TCP queries count was not always correct (#3288)
- A locking issue in exceedRespGen() might have caused crashes (#3277)
- Useless sockets were created in client mode (#3257)
- `addAnyTCRule()` was generating TC=1 responses even over TCP (#3251)

21.63.3 Web interface

- Cleanup of the HTML by Sander Hoentjen
- Fixed an XSS reported by @janeczku (#3217)
- Removed remote images
- Set the charset to UTF-8, added some security-related and CORS HTTP headers
- Added server latency by Jan Broer (#3201)
- Switched to official minified versions of JS scripts, by Sander Hoentjen (#3317)
- Don't log unauthenticated HTTP request as an authentication failure

21.63.4 Various documentation updates and minor cleanups:

- Added documentation for Advanced DNS Protection features (Dynamic rules, `maintenance()`)
- Make `topBandwidth()` default to the top 10 clients
- Replaced readline with libedit
- Added GPL2 License (#3200)
- Added incbin License (#3269)
- Updated completion rules
- Removed wrong option `--daemon-no` by Stefan Schmidt

21.64 1.0.0-alpha1

Released December 24th 2015

Initial release

CHAPTER

TWENTYTWO

UPGRADE GUIDE

22.1 1.8.x to 1.9.0

dnsdist now supports a new library for dealing with incoming DNS over HTTPS queries: `nghttp2`. The previously used library, `h2o`, can still be used but is now deprecated, disabled by default (see `--with-h2o` to enable it back) and will be removed in the future, as it is unfortunately no longer maintained in a way that is suitable for use as a library (see https://github.com/h2o/h2o/issues/3230). See the `library` parameter on the *`addDOHLocal()`* directive for more information on how to select the library used when dnsdist is built with support for both `h2o` and `nghttp2`. The default is now `nghttp2` whenever possible. Note that `nghttp2` only supports HTTP/2, and not HTTP/1, while `h2o` supported both. This is not an issue for actual DNS over HTTPS clients that support HTTP/2, but might be one in setups running dnsdist behind a reverse-proxy that does not support HTTP/2. See *DNS-over-HTTPS (DoH)* for some work-around.

SNMP support is no longer enabled by default during `configure`, requiring `--with-net-snmp` to be built.

The use of *`makeRule()`* is now deprecated, please use *`NetmaskGroupRule()`* or *`QNameSuffixRule()`* instead. Passing a string or list of strings instead of a *`DNSRule`* to these functions is deprecated as well, *`NetmaskGroupRule()`* and *`QNameSuffixRule()`* should there again be used instead:

- *`addAction()`*
- *`addResponseAction()`*
- *`addCacheHitResponseAction()`*
- *`addCacheInsertedResponseAction()`*
- *`addSelfAnsweredResponseAction()`*

22.2 1.7.x to 1.8.0

Responses to AXFR and IXFR queries are no longer cached.

Cache-hits are now counted as responses in our metrics.

Cache hits are now inserted into the in-memory ring buffers, while before 1.8.0 only cache misses were inserted. This has a very noticeable impact on *Dynamic Rule Generation* since cache hits now considered when computing the rcode rates and ratios, as well as the response bandwidth rate.

The *`setMaxTCPConnectionsPerClient()`* limit is now properly applied to DNS over HTTPS connections, in addition to DNS over TCP and DNS over TLS ones.

The configuration check will now fail if the configuration file does not exist. For this reason we now create a default configuration file, based on the file previously called `dnsdistconf.lua`, which contains commented-out examples of how to set up dnsdist.

Latency metrics have been broken down:

- per incoming protocol (Do53 UDP, Do53 TCP, DoT, DoH) for global latency metrics
- between UDP (Do53) and TCP (Do53 TCP, DoT, DoH) for backend latency metrics

22.3 1.7.0 to 1.7.1

In our Docker image, our binaries are no longer granted the `net_bind_service` capability, as this is unnecessary in many deployments. For more information, see the section "Privileged ports" in Docker-README. (This note was in the 1.6.x to 1.7.0 upgrade guide before, but the change was not present in 1.7.0.)

22.4 1.6.x to 1.7.0

Truncated responses received over UDP for DoH clients will now be retried over TCP.

setTCPUseSinglePipe() has been removed.

Unless set via *setMaxTCPClientThreads()* the number of TCP workers now defaults to 10, instead of the number of TCP binds.

Plain-text API keys and passwords for web server authentication are now strongly discouraged. The *hashPassword()* method can be used to generate a hashed and salted version of passwords and API keys instead, so that the plain-text version can no longer be found in either the configuration file or the memory of the running process.

22.5 1.5.x to 1.6.0

The packet cache no longer hashes EDNS Cookies by default, which means that two queries that are identical except for the content of their cookie will now be served the same answer. This only works if the backend is not returning any answer containing EDNS Cookies, otherwise the wrong cookie might be returned to a client. To prevent this, the `cookieHashing=true` parameter might be passed to *newPacketCache()* so that cookies are hashed, resulting in separate entries in the packet cache.

All TCP worker threads are now created at startup, instead of being created on-demand. The existing behaviour was useful for very small setups but did not scale quickly to a large amount of TCP connections. The new behaviour can cause a noticeable increase of TCP connections between dnsdist and its backends, as the TCP connections are not shared between TCP worker threads. This is especially true for setups with a large number of frontends (*addLocal()*, *addTLSLocal()*, and *addDNSCryptBind()* directives), as 1.6.0 sets the number of TCP workers to the number of TCP-enabled binds (with a minimum of 10), unless that number has been set explicitly via *setMaxTCPClientThreads()*.

Several actions have been renamed so that almost all actions that allow further processing of rules start with 'Set', to prevent mistakes:

- `DisableECSAction` to *SetDisableECSAction()*
- `DisableValidationAction` to *SetDisableValidationAction()*
- `ECSOverrideAction` to *SetECSOverrideAction()*
- `ECSPrefixLengthAction` to *SetECSPrefixLengthAction()*
- `MacAddrAction` to *SetMacAddrAction()*
- `NoRecurseAction` to *SetNoRecurseAction()*
- `SkipCacheAction` to *SetSkipCacheAction()*
- `TagAction` to *SetTagAction()*
- `TagResponseAction` to *SetTagResponseAction()*
- `TempFailureCacheTTLAction` to *SetAdditionalProxyProtocolValueAction()*
- `SetNegativeAndSOAAction` to *NegativeAndSOAAction()*

Some ambiguous commands have also been renamed to prevent mistakes:

- *topCacheHitResponseRule* to `mvCacheHitResponseRuleToTop()`
- *topResponseRule* to `mvResponseRuleToTop()`
- *topRule* to `mvRuleToTop()`
- *topSelfAnsweredResponseRule* to `mvSelfAnsweredResponseRuleToTop()`

The use of additional parameters on the `webserver()` command has been deprecated in favor of using `setWebserverConfig()`.

Regular users should not be impacted by this change, but packagers should be aware that since 1.6.0 dnsdist now uses the C++17 standard instead of the C++11 one it was previously using.

22.6 1.4.x to 1.5.0

DOH endpoints specified in the fourth parameter of `addDOHLocal()` are now specified as exact paths instead of path prefixes. The default endpoint also switched from `/` to `/dns-query`. For example, `addDOHLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/private/example.com.key', { "/dns-query" })` will now only accept queries for `/dns-query` and no longer for `/dns-query/foo/bar`. This change also impacts the HTTP response rules set via `DOHFrontend:setResponsesMap()`, since queries whose paths are not allowed will be discarded before the rules are evaluated. If you want to accept DoH queries on `/dns-query` and redirect `/rfc` to the DoH RFC, you need to list `/rfc` in the list of paths:

```
addDOHLocal('2001:db8:1:f00::1', '/etc/ssl/certs/example.com.pem', '/etc/ssl/
↪private/example.com.key', { '/dns-query', '/rfc'})
map = { newDOHResponseMapEntry("^/rfc$", 307, "https://www.rfc-editor.org/info/
↪rfc8484") }
dohFE = getDOHFrontend(0)
dohFE:setResponsesMap(map)
```

The systemd service-file that is installed no longer uses the `root` user to start. It uses the user and group set with the `--with-service-user` and `--with-service-group` switches during configuration, “dnsdist” by default. This could mean that dnsdist can no longer read its own configuration, or other data. It is therefore recommended to recursively `chown` directories used by dnsdist:

```
chown -R root:dnsdist /etc/dnsdist
```

Packages provided on the PowerDNS Repository will `chown` directories created by them accordingly in the post-installation steps.

This might not be sufficient if the dnsdist configuration refers to files outside of the /etc/dnsdist directory, like DoT or DoH certificates and private keys. Many ACME clients used to get and renew certificates, like CertBot, set permissions assuming that services are started as root. For that particular case, making a copy of the necessary files in the /etc/dnsdist directory is advised, using for example CertBot’s `--deploy-hook` feature to copy the files with the right permissions after a renewal.

The `webserver()` configuration now has an optional ACL parameter, that defaults to “127.0.0.1, ::1”.

22.7 1.3.x to 1.4.0

`addLuaAction()` and `addLuaResponseAction()` have been removed. Instead, use `addAction()` with a `LuaAction()`, or `addResponseAction()` with a `LuaResponseAction()`.

`newPacketCache()` now takes an optional table as its second argument, instead of several optional parameters.

Lua’s constants for DNS response codes and QTypes have been moved from the ‘dnsdist’ prefix to, respectively, the ‘DNSQType’ and ‘DNSRCode’ prefix.

To improve security, all ambient capabilities are now dropped after the startup phase, which might prevent launching the webserver on a privileged port at run-time, or impact some custom Lua code. In addition, systemd's sandboxing features are now determined at compile-time, resulting in more restrictions on recent distributions. See pull requests 7138 and 6634 for more information.

If you are compiling dnsdist, note that several ./configure options have been renamed to provide a more consistent experience. Features that depend on an external component have been prefixed with '–with-' while internal features use '–enable-'. This lead to the following changes:

- `--enable-fstrm` to `--enable-dnstap`
- `--enable-gnutls` to `--with-gnutls`
- `--enable-libsodium` to `--with-libsodium`
- `--enable-libssl` to `--with-libssl`
- `--enable-re2` to `--with-re2`

22.8 1.3.2 to 1.3.3

When upgrading from a package before 1.3.3, on CentOS 6 and RHEL 6, dnsdist will be stopped instead of restarted.

22.9 1.2.x to 1.3.x

In version 1.3.0, these things have changed.

The *Working with the dnsdist Console* has an ACL now, which is set to `{"127.0.0.0/8", "::1/128"}` by default. Add the appropriate *`setConsoleACL()`* and *`addConsoleACL()`* statements to the configuration file.

The `--daemon` option is removed from the **`dnsdist`** binary, meaning that **`dnsdist`** will not fork to the background anymore. Hence, it can only be run on the foreground or under a supervisor like systemd, supervisord and `daemon(8)`.

Due to changes in the architecture of **`dnsdist`**, several of the shortcut rules have been removed after deprecating them in 1.2.0. All removed functions have their equivalent *`addAction()`* listed. Please check the configuration for these statements (or use `dnsdist --check-config`) and update where needed. This removal affects these functions:

- `addAnyTCRule()`
- `addDelay()`
- `addDisableValidationRule()`
- `addDomainBlock()`
- `addDomainCNAMESpoof()`
- `addDomainSpoof()`
- `addNoRecurseRule()`
- `addPoolRule()`
- `addQPSLimit()`
- `addQPSPoolRule()`

22.10 1.1.0 to 1.2.0

In 1.2.0, several configuration options have been changed:

As the amount of possible settings for listen sockets is growing, all listen-related options must now be passed as a table as the second argument to both *addLocal()* and *setLocal()*. See the function's reference for more information.

The `BlockFilter` function is removed, as *addAction()* combined with a *DropAction()* can do the same.

CHAPTER

TWENTYTHREE

SECURITY ADVISORIES

All security advisories for the DNSDist are listed here.

23.1 PowerDNS Security Advisory 2017-01 for dnsdist: Crafted backend responses can cause a denial of service

- CVE: CVE-2016-7069
- Date: 2017-08-21
- Credit: Guido Vranken
- Affects: dnsdist up to and including 1.2.0 on 32-bit systems
- Not affected: dnsdist 1.2.0, dnsdist on 64-bit (all versions)
- Severity: Low
- Impact: Degraded service or Denial of service
- Exploit: This issue can be triggered by sending specially crafted response packets from a backend
- Risk of system compromise: No
- Solution: Upgrade to a non-affected version
- Workaround: Disable EDNS Client Subnet addition

An issue has been found in dnsdist in the way EDNS0 OPT records are handled when parsing responses from a backend. When dnsdist is configured to add EDNS Client Subnet to a query, the response may contain an EDNS0 OPT record that has to be removed before forwarding the response to the initial client. On a 32-bit system, the pointer arithmetic used when parsing the received response to remove that record might trigger an undefined behavior leading to a crash.

dnsdist up to and including 1.1.0 is affected on 32-bit systems. dnsdist 1.2.0 is not affected, dnsdist on 64-bit systems is not affected.

For those unable to upgrade to a new version, a minimal patch is available for 1.1.0

We would like to thank Guido Vranken for finding and subsequently reporting this issue.

23.2 PowerDNS Security Advisory 2017-02 for dnsdist: Alteration of ACLs via API authentication bypass

- CVE: CVE-2017-7557
- Date: 2017-08-21
- Credit: Nixu

- Affects: dnsdist 1.1.0
- Not affected: dnsdist 1.0.0, 1.2.0
- Severity: Low
- Impact: Access restriction bypass
- Exploit: This issue can be triggered by tricking an authenticated user into visiting a crafted website
- Risk of system compromise: No
- Solution: Upgrade to a non-affected version
- Workaround: Keep the API read-only (default) via setAPIWritable(false)

An issue has been found in dnsdist 1.1.0, in the API authentication mechanism. API methods should only be available to a user authenticated via an X-API-Key HTTP header, and not to a user authenticated on the webserver via Basic Authentication, but it was discovered by Nixu during a source code audit that dnsdist 1.1.0 allows access to all API methods to both kind of users.

In the default configuration, the API does not provide access to more information than the webserver does, and therefore this issue has no security implication. However if the API is allowed to make configuration changes, via the setAPIWritable(true) option, this allows a remote unauthenticated user to trick an authenticated user into editing dnsdist's ACLs by making him visit a crafted website containing a Cross-Site Request Forgery.

For those unable to upgrade to a new version, a minimal patch is available for 1.1.0

23.3 PowerDNS Security Advisory for dnsdist 2018-08: Record smuggling when adding ECS or XPF

- CVE: CVE-2018-14663
- Date: November 8th 2018
- Affects: PowerDNS DNSDist up to and including 1.3.2
- Not affected: 1.3.3
- Severity: Low
- Impact: Insufficient validation
- Exploit: This problem can be triggered via crafted queries
- Risk of system compromise: No
- Solution: Upgrade to a non-affected version

An issue has been found in PowerDNS DNSDist allowing a remote attacker to craft a DNS query with trailing data such that the addition of a record by dnsdist, for example an OPT record when adding EDNS Client Subnet, might result in the trailing data being smuggled to the backend as a valid record while not seen by dnsdist. This is an issue when dnsdist is deployed as a DNS Firewall and used to filter some records that should not be received by the backend. This issue occurs only when either the 'useClientSubnet' or the experimental 'addXPF' parameters are used when declaring a new backend.

This issue has been assigned CVE-2018-14663 by Red Hat.

PowerDNS DNSDist up to and including 1.3.2 is affected.

We would like to thank Richard Gibson for finding and subsequently reporting this issue.

If you have a security problem to report, please see our *security policy*.

CHAPTER

TWENTYFOUR

POWERDNS SECURITY POLICY

If you have a security problem to report, please email us at both peter.van.dijk@powerdns.com and remi.gacogne@powerdns.com. In case you want to encrypt your report using PGP, please use: https://doc.powerdns.com/powerdns-keyblock.asc

Please do not mail security issues to public lists, nor file a ticket, unless we do not get back to you in a timely manner. We fully credit reporters of security issues, and respond quickly, but please allow us a reasonable timeframe to coordinate a response.

We remind PowerDNS and dnsdist users that under the terms of the GNU General Public License, PowerDNS and dnsdist come with ABSOLUTELY NO WARRANTY. This *license* is included in this documentation.

If you believe you have found a security vulnerability that applies to DNS implementations generally, and you want to report this responsibly to a number of implementers, you might consider also using the Open Source DNS Vulnerability mailing list, managed by DNS-OARC.

24.1 YesWeHack

Security issues can also be reported on our YesWeHack page and might fetch a bounty. Do note that only the PowerDNS software is in scope for the YesWeHack program, not our websites or other infrastructure.

24.2 Disclosure Policy

- Let us know as soon as possible upon discovery of a potential security issue, and we'll make every effort to quickly resolve the issue.
- Provide us a reasonable amount of time to resolve the issue before any disclosure to the public or a third-party.
- We will always credit researchers in our *Security Advisories*.

CHAPTER TWENTYFIVE

GLOSSARY

ACL Access Control List

Open Resolver A recursive DNS server available for many hosts on the internet. Usually without adequate rate-limiting, allowing it to be used in reflection attacks.

QPS Queries Per Second

CHAPTER

TWENTYSIX

POWERDNS/DNSDIST LICENSE

We remind PowerDNS and dnsdist users that under the terms of the GNU General Public License, PowerDNS and dnsdist come with ABSOLUTELY NO WARRANTY.

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
```

(continues on next page)

(continued from previous page)

```
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
```

(continues on next page)

(continued from previous page)

```
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
```

(continues on next page)

(continued from previous page)

```
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.
```

(continues on next page)

(continued from previous page)

```
  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
```

(continues on next page)

(continued from previous page)

```
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

CHAPTER

TWENTYSEVEN

END OF LIFE STATEMENTS

We aim to have a major release every six months. The latest major release train receives correctness, stability and security updates by the way of minor releases. We support older releases with critical updates for one year after the following major release.

Older releases are marked end of life and receive no updates at all. Pre-releases do not receive immediate security updates.

The currently supported release train of PowerDNS DNSdist is 1.9.

PowerDNS DNSdist 1.8 will only receive critical updates and will be End of Life one year after PowerDNS DNSdist 1.9 was released.

PowerDNS DNSdist 1.7 will only receive critical updates and will be End of Life one year after PowerDNS DNSdist 1.8 was released.

Older versions are End of Life.

Note: Users with a commercial agreement with PowerDNS.COM BV or Open-Xchange can receive extended support for releases which are End Of Life. If you are such a user, these EOL statements do not apply to you. Please refer to the support commitment for details. Note that for the Open Source support channels we only support the latest minor release of a release train. That means that we ask you to reproduce potential issues on the latest minor release first.

Table 1: PowerDNS dnsdist Release Life Cycle

Version	Release date	Security-Only updates	End of Life
1.9	February 16 2024		
1.8	March 30 2023	February 16 2024	February 16 2025
1.7	January 17 2022	March 30 2023	March 30 2024
1.6	May 11 2021	March 30 2023	EOL (February 16 2024)
1.5	July 30 2020	January 17 2022	EOL (March 30 2023)
1.4	November 20 2019	May 2021	EOL (January 17 2022)
1.3	March 30 2018	EOL	EOL (May 2021)
1.2	August 21 2017	EOL	EOL
1.1	December 29 2016	EOL	EOL
1.0	April 21 2016	EOL	EOL

HTTP ROUTING TABLE

INDEX