dnsdist

PowerDNS.COM BV

Jan 15, 2021

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="2001:4860:4860::8888", qps=1})
newServer({address="2001:4860:4860::8844", qps=1})
newServer({address="2620:0:ccc::2", qps=10})
newServer({address="2620:0:ccd::2", name="dns1", qps=10})
newServer("192.168.1.2")
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++ 2011 support (like GCC 4.8+ or clang 3.5+) and GNU make are required. dnsdist depends on the following libraries:

- Boost
- Lua 5.1+ or LuaJit
- Editline (libedit)
- libsodium (optional)

- protobuf (optional, not needed as of 1.6.0)
- re2 (optional)

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:

- D630 0CAB CBF4 69BB E392 E503 A208 ED4F 8AF5 8446
- FBAE 0323 821C 7706 A5CA 151B DCF5 13FA 7EED 19F3
- 1628 90D0 689D D12D D33E 4696 1C5E E990 D2E7 1575
- B76C D467 1C09 68BA A87D E61C 5E50 715B F2FF E1A7
- 16E1 2866 B773 8C73 976A 5743 6FFC 3343 9B0D 04DF

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

- 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.

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 8.8.8.8 2001:4860:4860::8888
```

This will make dnsdist listen on IP address 127.0.0.1, port 5300 and forward all queries to the two 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="2001:4860:4860::8888", qps=1})
newServer({address="2001:4860:4860::8844", qps=1})
newServer({address="2620:0:ccc::2", qps=10})
newServer({address="2620:0:ccd::2", name="dns1", qps=10})
newServer("192.168.1.2")
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 [2001:4860:4860::8888]:53 as 'up'
Marking downstream [2001:4860:4860::8844]:53 as 'up'
Marking downstream [2620:0:ccc::2]:53 as 'up'
Marking downstream [2620:0:ccd::2]:53 as 'up'
Marking downstream 192.168.1.2:53 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   [2001:4860:4860::8888]:53    up       0.0       1   1  1          1       0   0.
↪0   0.0
1   [2001:4860:4860::8844]:53    up       0.0       1   1  1          0       0   0.
↪0   0.0
2   [2620:0:ccc::2]:53           up       0.0      10   1  1          0       0   0.
↪0   0.0
3   [2620:0:ccd::2]:53           up       0.0      10   1  1          0       0   0.
↪0   0.0
4   192.168.1.2:53               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   [2001:4860:4860::8888]:53    up       1.0       1   1  1          7       0   0.
↪0   1.6
1   [2001:4860:4860::8844]:53    up       1.0       1   1  1          6       0   0.
↪0   0.6
2   [2620:0:ccc::2]:53           up      10.3      10   1  1         64       0   0.
↪0   2.4
3   [2620:0:ccd::2]:53           up      10.3      10   1  1         63       0   0.
↪0   2.4
4   192.168.1.2:53               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   [2001:4860:4860::8888]: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 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('2.0.0.0/8')
-- Wed Feb 22 2017 12:03:50 CET
addACL('2.0.0.0/8')
-- Wed Feb 22 2017 12:03:51 CET
addACL('2.0.0.0/8')
-- 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.

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.

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.

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), NoRecurseAction())
```

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()`*.

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.

5.2 Rule Generators

`dnsdist` contains several functions that make it easier to add actions and rules.

`addAnyTCRule`()
Deprecated since version 1.2.0.

Set the TC-bit (truncate) on ANY queries received over UDP, forcing a retry over TCP. This function has been deprecated as of 1.2.0 and removed in 1.3.0. This is equivalent to doing:

```
addAction(AndRule({QTypeRule(DNSQType.ANY), TCPRule(false)}), TCAction())
```

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

`addDelay`(*DNSrule*, *delay*)
Deprecated since version 1.2.0.

Delay the query for `delay` milliseconds before sending to a backend. This function has been deprecated as of 1.2.0 and removed in 1.3.0, please use instead:

addAction(DNSRule, DelayAction(delay))

Parameters

- **`DNSRule`** – The DNSRule to match traffic
- **`delay`** (*int*) – The delay time in milliseconds.

`addDisableValidationRule`(*DNSrule*)
Deprecated since version 1.2.0.

Set the CD (Checking Disabled) flag to 1 for all queries matching the DNSRule. This function has been deprecated as of 1.2.0 and removed in 1.3.0. Please use the *`DisableValidationAction()`* action instead.

`addDomainBlock`(*domain*)
Deprecated since version 1.2.0.

Drop all queries for `domain` and all names below it. Deprecated as of 1.2.0 and will be removed in 1.3.0, please use instead:

addAction(domain, DropAction())

Parameters **domain** (*string*) – The domain name to block

addDomainSpoof (*domain*, *IPv4*[, *IPv6*])
addDomainSpoof (*domain, {IP[,...]}*)
Deprecated since version 1.2.0.

Generate answers for A/AAAA/ANY queries. This function has been deprecated as of 1.2.0 and removed in 1.3.0, please use:

addAction(domain, SpoofAction({IP[,...]}))

or:

addAction(domain, SpoofAction(IPv4[, IPv6]))

Parameters

- **domain** (*string*) – Domain name to spoof for
- **IPv4** (*string*) – IPv4 address to spoof in the reply
- **IPv6** (*string*) – IPv6 address to spoof in the reply
- **IP** (*string*) – IP address to spoof in the reply

addDomainCNAMESpoof (*domain*, *cname*)
Deprecated since version 1.2.0.

Generate CNAME answers for queries. This function has been deprecated as of 1.2.0 and removed in 1.3.0, in favor of using:

addAction(domain, SpoofCNAMEAction(cname))

Parameters

- **domain** (*string*) – Domain name to spoof for
- **cname** (*string*) – Domain name to add CNAME to

addLuaAction (*DNSrule*, *function*[, *options*])
Changed in version 1.3.0: Added the optional parameter `options`.

Changed in version 1.3.0: The second argument returned by the `function` can be omitted. For earlier releases, simply return an empty string.

Deprecated since version 1.4.0: Removed in 1.4.0, use *LuaAction()* with *addAction()* instead.

Invoke a Lua function that accepts a *DNSQuestion*. This function works similar to using *LuaAction()*. The `function` should return both a *DNSAction* and its argument *rule*. The *rule* is used as an argument of the following *DNSAction*: *DNSAction.Spoof*, *DNSAction.Pool* and *DNSAction.Delay*. If the Lua code fails, ServFail is returned.

Parameters

- **DNSRule** – match queries based on this rule
- **function** (*string*) – the name of a Lua function
- **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.

```
function luaaction(dq)
  if(dq.qtype==DNSQType.NAPTR)
  then
    return DNSAction.Pool, "abuse" -- send to abuse pool
  else
    return DNSAction.None, ""      -- no action
    -- return DNSAction.None       -- as of dnsdist version 1.3.0
  end
end

addLuaAction(AllRule(), luaaction)
```

addLuaResponseAction(*DNSrule*, *function*[, *options*])

Changed in version 1.3.0: Added the optional parameter `options`.

Changed in version 1.3.0: The second argument returned by the `function` can be omitted. For earlier releases, simply return an empty string.

Deprecated since version 1.4.0: Removed in 1.4.0, use *LuaResponseAction()* with *addResponseAction()* instead.

Invoke a Lua function that accepts a *DNSResponse*. This function works similar to using *LuaResponseAction()*. The `function` should return both a *DNSResponseAction* and its argument *rule*. The *rule* is used as an argument of the *DNSResponseAction.Delay*. If the Lua code fails, ServFail is returned.

Parameters

- **DNSRule** – match queries based on this rule
- **function** (*string*) – the name of a Lua function
- **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.

addNoRecurseRule(*DNSrule*)

Deprecated since version 1.2.0.

Clear the RD flag for all queries matching the rule. This function has been deprecated as of 1.2.0 and removed in 1.3.0, please use:

addAction(DNSRule, NoRecurseAction())

Parameters **DNSRule** – match queries based on this rule

addPoolRule(*DNSRule*, *pool*)

Deprecated since version 1.2.0.

Send queries matching the first argument to the pool `pool`. e.g.:

```
addPoolRule("example.com", "myPool")
```

This function has been deprecated as of 1.2.0 and removed in 1.3.0, this is equivalent to:

```
addAction("example.com", PoolAction("myPool"))
```

Parameters

- **DNSRule** – match queries based on this rule
- **pool** (*string*) – The name of the pool to send the queries to

addQPSLimit (*DNSrule*, *limit*)
Deprecated since version 1.2.0.

Limit queries matching the DNSRule to `limit` queries per second. All queries over the limit are dropped. This function has been deprecated as of 1.2.0 and removed in 1.3.0, please use:

addAction(DNSRule, QPSAction(limit))

Parameters

- **DNSRule** – match queries based on this rule
- **limit** (*int*) – QPS limit for this rule

addQPSPoolRule (*DNSRule*, *limit*, *pool*)
Deprecated since version 1.2.0.

Send at most `limit` queries/s for this pool, letting the subsequent rules apply otherwise. This function has been deprecated as of 1.2.0 and removed in 1.3.0, as it is only a convenience function for the following syntax:

```
addAction("192.0.2.0/24", QPSPoolAction(15, "myPool")
```

Parameters

- **DNSRule** – match queries based on this rule
- **limit** (*int*) – QPS limit for this rule
- **pool** (*string*) – The name of the pool to send the queries to

5.3 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
```

For Rules related to the incoming query:

addAction (*DNSrule*, *action*[, *options*])
Changed in version 1.3.0: Added the optional parameter `options`.

Changed in version 1.6.0: Added `name` to the `options`.

Add a Rule and Action to the existing rules.

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.

clearRules()
Remove all current rules.

getAction(*n*) → Action
Returns the Action associated with rule `n`.

Parameters **n** (*int*) – The rule number

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()*.

newRuleAction(*rule*, *action*[, *options*])
Changed in version 1.3.0: Added the optional parameter `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 *Matching Packets (Selectors)*)
- **action** (*Action*) – The Action (see *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.

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*])
Changed in version 1.3.0: `options` optional parameter added

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 1.3.0: `id` can now be an UUID.

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

For Rules related to responses:

addResponseAction(*DNSRule*, *action*[, *options*])

Changed in version 1.3.0: Added the optional parameter `options`.

Changed in version 1.6.0: Added `name` to the `options`.

Add a Rule and Action for responses to the existing rules.

Parameters

- **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.

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 1.3.0: `id` can now be an UUID.

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*])

Changed in version 1.3.0: `options` optional parameter added

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()*.

Functions for manipulating Cache Hit Response Rules:

addCacheHitResponseAction(*DNSRule*, *action*[, *options*])
New in version 1.2.0.

Changed in version 1.3.0: Added the optional parameter `options`.

Changed in version 1.6.0: Added `name` to the `options`.

Add a Rule and ResponseAction for Cache Hits to the existing rules.

Parameters

- **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.

mvCacheHitResponseRule(*from*, *to*)
New in version 1.2.0.

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*)
New in version 1.2.0.

Changed in version 1.3.0: `id` can now be an UUID.

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*])
New in version 1.2.0.

Changed in version 1.3.0: `options` optional parameter added

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()
New in version 1.2.0.

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()*.

Functions for manipulating Self-Answered Response Rules:

addSelfAnsweredResponseAction(*DNSRule*, *action*[, *options*])
New in version 1.3.0.

Changed in version 1.6.0: Added `name` to the `options`.

Add a Rule and Action for Self-Answered queries to the existing rules.

Parameters

- **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.

mvSelfAnsweredResponseRule(*from*, *to*)
New in version 1.3.0.

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*)
New in version 1.3.0.

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*])
New in version 1.3.0.

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()
New in version 1.3.0.

Changed in version 1.6.0: Replaced by *mvSelfAnsweredResponseRuleToTop()*

Before 1.6.0 this function used to move the last cache hit response rule to the first position, which is now handled by *mvSelfAnsweredResponseRuleToTop()*.

Move the last self answered response rule to the first position.

5.4 Matching Packets (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`

New in version 1.2.0: A DNSRule can also be a *DNSName* or a list of these

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.

Matches DNS over HTTPS queries with a HTTP header `name` whose content matches the regular expression `regex`.

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

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*]]]]]])
Changed in version 1.3.1: Added the optional parameters `expiration`, `cleanupDelay` and `scanFraction`.

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%

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

Matches traffic from/to the network range specified in `nmg`.

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 NetMaskGroup to match on
- **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

ProbaRule(*probability*)
New in version 1.3.0.

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*)
New in version 1.2.0: Matches queries with the specified qname exactly.

param string qname Qname to match

QNameSetRule(*set*)
New in version 1.4.0: Matches if the set contains exact qname.

To match subdomain names, see *SuffixMatchNodeRule()*.

param DNSNameSet set Set with qnames.

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()
New in version 1.2.0.

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.

Parameters **name** (*str*) – The exact SNI name to match.

SuffixMatchNodeRule(*smn*[, *quiet*])
Matches based on a group of domain suffixes for rapid testing of membership. Pass true as second parameter to prevent listing of all domains matched.

To match domain names exactly, see *QNameSetRule()*.

Parameters

- **smn** (SuffixMatchNode) – The SuffixMatchNode to match on
- **quiet** (*bool*) – Do not display the list of matched domains in Rules. Default is false.

TagRule(*name*[, *value*])
New in version 1.3.0.

Matches question or answer with a tag named `name` set. If `value` is specified, the existing tag value should match too.

Parameters

- **name** (*bool*) – The name of the tag that has to be set
- **value** (*bool*) – 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*)
New in version 1.3.3.

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

5.4.1 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 the `selectors` Rules does match.

Parameters **selector** (*{Rule}*) – A table of Rules

5.4.2 Convenience Functions

makeRule(*rule*)
Make a *NetmaskGroupRule()* or a *SuffixMatchNodeRule()*, depending on it is called. `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 to convert to a rule.

5.5 Actions

Matching Packets (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. The following actions exist.

AddProxyProtocolValueAction(*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.

Parameters

- **type** (*int*) – The type of the value to send, ranging from 0 to 255 (both included)
- **value** (*str*) – The binary-safe value

AllowAction()
Let these packets go through.

AllowResponseAction()
Let these packets go through.

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()
Disable the sending of ECS to the backend. Subsequent rules are processed after this action.

DisableValidationAction()
Set the CD bit in the query and let it go through.

DnstapLogAction(*identity*, *logger*[, *alterFunction*])
New in version 1.3.0.

Send the 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*])
New in version 1.3.0.

Send the 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*)
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*)
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()*).

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

LogAction([*filename*[, *binary*[, *append*[, *buffered*[, *verboseOnly*[, *includeTimestamp*]]]]]])

Changed in version 1.4.0: Added the optional parameters `verboseOnly` and `includeTimestamp`, made `filename` optional.

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.

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.

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.

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

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*)
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

NoneAction ()
Does nothing. Subsequent rules are processed after this action.

NoRecurseAction ()
Strip RD bit from the question, let it go through. Subsequent rules are processed after this action.

PoolAction (*poolname*)
Send the packet into the specified pool.

Parameters **poolname** (*string*) – The name of the pool

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*)
Send the packet into the specified pool only if it does not exceed the `maxqps` queries per second limits. Letting the subsequent rules apply otherwise.

Parameters

- **maxqps** (*int*) – The QPS limit for that pool

- **poolname** (*string*) – The name of the pool

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*]])
Changed in version 1.3.0: `options` optional parameter added.

Changed in version 1.4.0: `ipEncryptKey` 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. 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.

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.

RemoteLogResponseAction(*remoteLogger*[, *alterFunction*[, *includeCNAME*[, *options*]]])
Changed in version 1.3.0: `options` optional parameter added.

Changed in version 1.4.0: `ipEncryptKey` 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. 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.

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.

SetECSAction(*v4*[, *v6*])
New in version 1.3.1.

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

SetNegativeAndSOAAction(*nxd*, *zone*, *ttl*, *mname*, *rname*, *serial*, *refresh*, *retry*, *expire*, *minimum*[, *options*])
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
- **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`.

Parameters **values** (*table*) – A table of types and values to send, for example: `{ [0] = foo", [42] = "bar" }`

SkipCacheAction()
Don't lookup the cache for this query, don't store the answer.

SkipCacheResponseAction()
New in version 1.6.0.

Don't store this answer into the cache.

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*])
New in version 1.5.0.

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 a "aaa" "bbb" TXT record:
addAction(AndRule({QNameRule('raw.powerdns.com.'), QTypeRule(DNSQType.TXT)}),␣
↪SpoofRawAction("\003aaa\004bbbb"))
-- 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"))
```

DNSName:toDNSString() is convenient for converting names to wire format for passing to SpoofRawAction.

Parameters

- **rawAnswer** (*string*) – The raw record data
- **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.

TagAction (*name*, *value*)
New in version 1.3.0.

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*)
New in version 1.3.0.

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 ()
Create answer to query with TC and RD bits set, to force the client to TCP.

TeeAction(*remote*[, *addECS*])
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.

Parameters

- **remote** (*string*) – An IP:PORT combination to send the copied queries to
- **addECS** (*bool*) – Whether or not to add ECS information. Default false

TempFailureCacheTTLAction(*ttl*)
Set the cache TTL to use for ServFail and Refused replies. TTL is not applied for successful replies.

Parameters **ttl** (*int*) – Cache TTL for temporary failure replies

CHAPTER

SIX

STATISTICS

dnsdist keeps statistics on the queries is 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*)
- 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 + 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.

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 downstream-send-errors

Number of errors when sending a query to a backend.

6.9 downstream-timeouts

Number of queries not answer in time by a backend.

6.10 dyn-block-nmg-size

Number of dynamic blocks entries.

6.11 dyn-blocked

Number of queries dropped because of a dynamic block.

6.12 empty-queries

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

6.13 fd-usage

Number of currently used file descriptors.

6.14 frontend-noerror

Number of NoError answers sent to clients.

6.15 frontend-nxdomain

Number of NXDomain answers sent to clients.

6.16 frontend-servfail

Number of ServFail answers sent to clients.

6.17 latency-avg100

Average response latency in microseconds of the last 100 packets

6.18 latency-avg1000

Average response latency in microseconds of the last 1000 packets.

6.19 latency-avg10000

Average response latency in microseconds of the last 10000 packets.

6.20 latency-avg1000000

Average response latency in microseconds of the last 1000000 packets.

6.21 latency-slow

Number of queries answered in more than 1 second.

6.22 latency-sum

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

6.23 latency-count

Number of queries contributing to response time histogram

6.24 latency-bucket

Number of queries contributing to response time histogram per latency bucket

6.25 latency0-1

Number of queries answered in less than 1 ms.

6.26 latency1-10

Number of queries answered in 1-10 ms.

6.27 latency10-50

Number of queries answered in 10-50 ms.

6.28 latency50-100

Number of queries answered in 50-100 ms.

6.29 latency100-1000

Number of queries answered in 100-1000 ms.

6.30 no-policy

Number of queries dropped because no server was available.

6.31 noncompliant-queries

Number of queries dropped as non-compliant.

6.32 noncompliant-responses

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

6.33 queries

Number of received queries.

6.34 rdqueries

Number of received queries with the recursion desired bit set.

6.35 real-memory-usage

Current memory usage.

6.36 responses

Number of responses received from backends. Note! This is not the number of responses sent to clients. To get that number, add 'cache-hits' and 'responses'.

6.37 rule-drop

Number of queries dropped because of a rule.

6.38 rule-nxdomain

Number of NXDomain answers returned because of a rule.

6.39 rule-refused

Number of Refused answers returned because of a rule.

6.40 rule-servfail

Number of ServFail answers returned because of a rule.

6.41 security-status

New in version 1.3.4.

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)

6.42 self-answered

Number of self-answered responses.

6.43 servfail-responses

Number of servfail answers received from backends.

6.44 trunc-failures

Number of errors encountered while truncating an answer.

6.45 udp-in-errors

New in version 1.5.0.

From /proc/net/snmp InErrors.

6.46 udp-noport-errors

New in version 1.5.0.

From /proc/net/snmp NoPorts.

6.47 udp-recvbuf-errors

New in version 1.5.0.

From /proc/net/snmp RcvbufErrors.

6.48 udp-sndbuf-errors

New in version 1.5.0.

From /proc/net/snmp SndbufErrors.

6.49 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 libsodium support enabled.

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

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

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-HTTPS (DOH)

dnsdist supports DNS-over-HTTPS (DoH, standardized in RFC 8484). To see if the installation supports this, run `dnsdist --version`. If the output shows `dns-over-https (DOH)`, DNS-over-HTTPS is supported.

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\\""}})
```

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)
```

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 })
```

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.

CHAPTER

ELEVEN

DNS-OVER-TLS

Since version 1.3.0, **dnsdist** supports experimental DNS-over-TLS support. 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.

CHAPTER

TWELVE

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

THIRTEEN

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.

13.1 Healthcheck

dnsdist uses a health check, sent once every second, to determine the availability of a backend server.

By default, an A query for "a.root-servers.net." is sent. A different query type, class and target can be specified by passing, respectively, the `checkType`, `checkClass` and `checkName` parameters to *`newServer()`*.

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

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

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

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})
```

13.2 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*.

CHAPTER

FOURTEEN

DYNAMIC RULE GENERATION

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:

```
function maintenance()
    addDynBlocks(exceedQRate(20, 10), "Exceeded query rate", 60)
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.

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.

14.1 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.

For example, instead of having something like:

```
function maintenance()
  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 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
```

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 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.

CHAPTER

FIFTEEN

GUIDES

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

15.1 Built-in webserver

To visually interact with dnsdist, try add *webserver()* to the configuration:

```
webserver("127.0.0.1:8083", "supersecretpassword", "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:

```
webserver("127.0.0.1:8083", "supersecretpassword", "supersecretAPIkey", {}, "192.0.
↪2.0/24, !192.0.2.1")
```

15.1.1 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 *webserver()*. For example, to remove the X-Frame-Options header and add a X-Custom one:

```
webserver("127.0.0.1:8080", "supersecret", "apikey", {["X-Frame-Options"]= "", ["X-
↪Custom"]="custom"})
```

Credentials can be changed over time using the *setWebserverConfig()* function.

15.1.2 dnsdist API

To access the API, the *apikey* must be set in the *webserver()* 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'
Content-Type: application/json
X-Content-Type-Options: nosniff
X-Frame-Options: deny
```

(continues on next page)

(continued from previous page)

```
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_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
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_self_answered Number of self-answered responses
# TYPE dnsdist_self_answered counter
dnsdist_self_answered 0
```

(continues on next page)

(continued from previous page)

```
# 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
# 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_uptime Uptime of the dnsdist process in seconds
# TYPE dnsdist_uptime gauge
dnsdist_uptime 39
# HELP dnsdist_real_memory_usage Current memory usage in bytes
# TYPE dnsdist_real_memory_usage gauge
dnsdist_real_memory_usage 10276864
# HELP dnsdist_noncompliant_queries Number of queries dropped as non-
↪compliant
# TYPE dnsdist_noncompliant_queries counter
dnsdist_noncompliant_queries 0
```

(continues on next page)

(continued from previous page)

```
# 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_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_user_msec Milliseconds spent by dnsdist in the user␣
↪state
# TYPE dnsdist_cpu_user_msec counter
dnsdist_cpu_user_msec 28
# HELP dnsdist_cpu_sys_msec Milliseconds spent by dnsdist in the system␣
↪state
# TYPE dnsdist_cpu_sys_msec counter
dnsdist_cpu_sys_msec 32
# HELP dnsdist_fd_usage Number of currently used file descriptors
# TYPE dnsdist_fd_usage gauge
dnsdist_fd_usage 17
# 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
dnsdist_server_queries{server="1_1_1_1",address="1.1.1.1:53"} 0
dnsdist_server_drops{server="1_1_1_1",address="1.1.1.1:53"} 0
dnsdist_server_latency{server="1_1_1_1",address="1.1.1.1:53"} 0
dnsdist_server_senderrors{server="1_1_1_1",address="1.1.1.1:53"} 0
dnsdist_server_outstanding{server="1_1_1_1",address="1.1.1.1:53"} 0
dnsdist_server_order{server="1_1_1_1",address="1.1.1.1:53"} 1
dnsdist_server_weight{server="1_1_1_1",address="1.1.1.1:53"} 1
dnsdist_server_queries{server="1_0_0_1",address="1.0.0.1:53"} 0
dnsdist_server_drops{server="1_0_0_1",address="1.0.0.1:53"} 0
dnsdist_server_latency{server="1_0_0_1",address="1.0.0.1:53"} 0
dnsdist_server_senderrors{server="1_0_0_1",address="1.0.0.1:53"} 0
dnsdist_server_outstanding{server="1_0_0_1",address="1.0.0.1:53"} 0
dnsdist_server_order{server="1_0_0_1",address="1.0.0.1:53"} 1
dnsdist_server_weight{server="1_0_0_1",address="1.0.0.1:53"} 2
dnsdist_frontend_queries{frontend="127.0.0.1:1153",proto="udp"} 0
dnsdist_frontend_queries{frontend="127.0.0.1:1153",proto="tcp"} 0
dnsdist_pool_servers{pool="_default_"} 2
dnsdist_pool_cache_size{pool="_default_"} 200000
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
```

(continues on next page)

(continued from previous page)

```
dnsdist_pool_cache_insert_collisions{pool="_default_"} 0
dnsdist_pool_cache_ttl_too_shorts{pool="_default_"} 0
```

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:
  username: dontcare
  password: yoursecret
```

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
```

(continues on next page)

(continued from previous page)

```
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": [
        "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"
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

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

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
- **queries** (*integer*) – The number of received queries on this bind
- **udp** (*boolean*) – true if this is a UDP bind
- **tcp** (*boolean*) – true if this is a TCP bind

Pool

A description of a pool of backend servers.

Object Properties

- **id** (*integer*) – Internal identifier
- **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
- **cacheLookupCollisions** (*integer*) – The number of times an entry retrieved from the cache based on the query hash did not match the actual query
- **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
- **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
- **id** (*integer*) – The position 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.")
- **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.

Changed in version 1.3.1: The `dropRate` property was added

Object Properties

- **address** (*string*) – The remote IP and port
- **id** (*integer*) – Internal identifier
- **latency** (*integer*) – The current latency of this backend server

- **name** (*string*) – The name of this server
- **order** (*integer*) – Order number
- **outstanding** (*integer*) – Number of currently outstanding queries
- **pools** (*[string]*) – The pools this server belongs to
- **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
- **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")
- **weight** (*integer*) – The weight assigned to this server
- **dropRate** (*float*) – The amount of packets dropped per second by this server

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

15.2 Server pools

dnsdist has the concept to "server pools", any number of servers can belong to a group.

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")
```

15.3 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.

15.3.1 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.

15.3.2 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)
```

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
]])
```

15.3.3 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.

15.3.4 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.

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.

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

15.4 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.

15.4.1 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.

15.4.2 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.

15.4.3 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.

CHAPTER

SIXTEEN

ADVANCED TOPICS

These chapters contain information on the advanced features of dnsdist

16.1 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.

16.1.1 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.

16.1.2 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')
```

16.2 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
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.

16.3 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
```

(continues on next page)

(continued from previous page)

```
  else
    return DNSAction.None, ""      -- no action
  end
end

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

16.4 Runtime-modifiable IP address sets

New in version 1.2.0.

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, ""
```

(continues on next page)

(continued from previous page)

```
        end
end

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

16.5 Using EDNS Client Subnet

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

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

- calling *DisableECSAction()* or setting `dq.useECS` to `false` prevents the sending of the ECS option.
- calling *ECSOverrideAction()* or setting `dq.ecsOverride` will override the global *setECSOverride()* value.
- calling `ECSPrefixLengthAction(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 *DisableECSAction()* 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.

16.6 Using XPF

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 experimental XPF record (from draft-bellis-dnsop-xpf) 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.

The XPF record 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.

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())
```

16.7 Using the Proxy Protocol

In order to provide the downstream server with the address of the real client, or at least the one talking to 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. This header contains 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 Such a Proxy Protocol header can also be passed from the client to dnsdist, using *setProxyProtocolACL()* to specify which clients to accept it from. 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()*, *AddProxyProtocolValueAction()* 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.

16.8 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), NoRecurseAction())
```

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()*.

16.9 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`). This 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`.

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 BPF filter can be used to block incoming queries manually:

```
> bpf = newBPFFilter(1024, 1024, 1024)
> bpf:attachToAllBinds()
> bpf:block(newCA("2001:DB8::42"))
> bpf:blockQName(newDNSName("evildomain.com"), 255)
> 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, but it's also possible to define a default BPF filter at configuration time, so it's automatically attached to every bind:

```
bpf = newBPFFilter(1024, 1024, 1024)
setDefaultBPFFilter(bpf)
```

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

```
> bpf = newBPFFilter(1024, 1024, 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(1024, 1024, 1024)
setDefaultBPFFilter(bpf)
dbpf = newDynBPFFilter(bpf)
function maintenance()
        addBPFFilterDynBlocks(exceedQRate(20, 10), dbpf, 60)
        dbpf:purgeExpired()
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.

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 1.6.0, the default BPF filter set via *setDefaultBPFFilter()* will automatically get used when a dynamic block is inserted via a *DynBlockRulesGroup*.

This feature has been successfully tested on Arch Linux, Arch Linux ARM, Fedora Core 23 and Ubuntu Xenial

16.10 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 threads
- Each backend has its own thread listening for UDP responses
- 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
- A carbon thread exports statistics to carbon servers if needed
- One or more webserver threads handle queries to the internal webserver

The maximum number of threads in the TCP pool is controlled by the *setMaxTCPClientThreads()* directive, and defaults to 10. This number can be increased to handle a large number of simultaneous TCP connections. If all the TCP threads are busy, new TCP connections are queued while they wait to be picked up. 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.

The maximum number of queued connections can be configured with *setMaxTCPQueuedConnections()* and defaults to 1000. 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.

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 10240 (65535 since 1.4.0), with a maximum value of 65535. Large installations are advised to increase the default value at the cost of a slightly increased memory usage.

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 *addLuaAction()*
- 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.

dnsdist design choices mean that the processing of UDP queries is done by only one thread per local bind. 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. This is especially true for Lua-intensive setups, because Lua processing in dnsdist is serialized by a unique lock for all threads.

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"})
```

16.11 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;

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 }
```

(continues on next page)

(continued from previous page)

```
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
    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
```

(continues on next page)

(continued from previous page)

```
    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 queries answered in less than 1 ms"
    ::= { stats 14 }

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
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
        "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
```

(continues on next page)

(continued from previous page)

```
        "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
    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
```

(continues on next page)

(continued from previous page)

```
    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 }

specialMemoryUsage OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION
        "Memory usage (more precise but expensive to retrieve)"
    ::= { stats 38 }

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 }

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,
```

(continues on next page)

(continued from previous page)

```
    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
    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
```

(continues on next page)

(continued from previous page)

```
    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
    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 {
```

(continues on next page)

(continued from previous page)

```
                    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)
                }

---
--- 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"
```

(continues on next page)

(continued from previous page)

```
    ::= { 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
    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 }

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

actionTrap NOTIFICATION-TYPE
```

(continues on next page)

(continued from previous page)

```
    OBJECTS {
        socketFamily,
        socketProtocol,
        fromAddress,
        toAddress,
        queryType,
        querySize,
        queryID,
        qName,
        qClass,
        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 }

---
--- 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,
        selfAnswered,
        downstreamTimeouts,
        downstreamSendErrors,
        truncFailures,
        noPolicy,
        latency01,
        latency110,
        latency1050,
        latency50100,
        latency1001000,
        latencySlow,
```

(continues on next page)

(continued from previous page)

```
        latencyAVG100,
        latencyAVG1000,
        latencyAVG10000,
        latencyAVG1000000,
        uptime,
        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
```

16.12 AXFR, IXFR and NOTIFY

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

- If the master 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 master 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 master:

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

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.

When **dnsdist** is deployed in front of slaves, however, an issue might arise with NOTIFY queries, because the slave will receive a notification coming from the **dnsdist** address, and not the master's one. One way to fix this issue is to allow NOTIFY from the **dnsdist** address on the slave 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.

16.13 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`.

16.13.1 Using systemd

New in version 1.3.0.

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`.

16.14 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 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.

CHAPTER

SEVENTEEN

REFERENCE GUIDES

These chapters contain extensive information on all functions and object available in dnsdist.

17.1 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".

17.1.1 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.

17.1.2 Global configuration

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.

Set the syslog logging facility to `facility`.

Parameters **facility** (*int*) – The new facility as a numeric value. Defaults to LOG_DAEMON.

Listen Sockets

addLocal(*address*[, *options*])
New in version 1.2.0.

Changed in version 1.3.0: Added `cpus` to the 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` parameter.

Add to the list of listen addresses.

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.

```
addLocal('0.0.0.0:5300', { reusePort=true })
```

This will bind to both UDP and TCP on port 5300 with SO_REUSEPORT enabled.

addLocal(*address*[[[, *do_tcp*], *so_reuseport*], *tcp_fast_open_qsize*])
Deprecated since version 1.2.0.

Add to the list of addresses listened on.

Parameters

- **address** (*str*) – The IP Address with an optional port to listen on. The default port is 53.
- **do_tcp** (*bool*) – Also bind a TCP port on `address`, defaults to true.
- **so_reuseport** (*bool*) – Use `SO_REUSEPORT` if it is available, defaults to false
- **tcp_fast_open_qsize** (*int*) – The size of the TCP Fast Open queue. Set to a number higher than 0 to enable TCP Fast Open when available. Default is 0.

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 `/`. Added `tcpListenQueueSize` parameter.

Listen on the specified address and TCP port for incoming DNS over HTTPS connections, presenting the specified X.509 certificate. If no certificate (or key) files are specified, listen for incoming DNS over HTTP connections instead.

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, 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.
- **urls** (*str-or-list*) – The path part of a URL, or a list of paths, to accept queries on. Any query with a path under one of these will be treated as a DoH query. 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".
- `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.
- `ticketsKeysRotationDelay`: int - Set the delay before the TLS tickets key is rotated, in seconds. Default is 43200 (12h).
- `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.

`addTLSLocal` (*address*, *certFile(s)*, *keyFile(s)*[, *options*])

New in version 1.3.0.

Changed in version 1.3.1: `certFile(s)` and `keyFile(s)` parameters accept a list of files. `sessionTickets` option added.

Changed in version 1.3.3: `numberOfStoredSessions` option added.

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: Added `maxInFlight` parameter.

Listen on the specified address and TCP port for incoming DNS over TLS connections, presenting the specified X.509 certificate.

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, 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 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.
- `ticketsKeysRotationDelay`: int - Set the delay before the TLS tickets key is rotated, in seconds. Default is 43200 (12h).
- `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.

setLocal (*address*[, *options*])
New in version 1.2.0.

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()*.

setLocal (*address*[[[, *do_tcp*], *so_reuseport*], *tcp_fast_open_qsize*])
Deprecated since version 1.2.0.

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.
- **do_tcp** (*bool*) – Also bind a TCP port on `address`, defaults to true.
- **so_reuseport** (*bool*) – Use `SO_REUSEPORT` if it is available, defaults to false
- **tcp_fast_open_qsize** (*int*) – The size of the TCP Fast Open queue. Set to a number higher than 0 to enable TCP Fast Open when available. Default is 0.

Control Socket, Console and Webserver

addConsoleACL (*netmask*)
New in version 1.3.0.

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 sub-netmask, 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*)
New in version 1.2.0.

Whether to log the opening and closing of console connections.

Parameters **enabled** (*bool*) – Default to true.

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*)
New in version 1.3.0.

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*)
New in version 1.3.3.

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

webserver(*listen_address*, *password*[, *apikey*[, *custom_headers*[, *acl*]]])
Changed in version 1.5.0: `acl` optional parameter added.

Launch the *Built-in webserver* with statistics and the API.

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
- **custom_headers** (*{[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*)
New in version 1.3.3.

Changed in version 1.5.0: `acl` optional parameter added.

Setup webserver configuration. See *webserver()*.

Parameters **options** (*table*) – A table with key: value pairs with webserver options.

Options:

- `password=newPassword`: string - Changes the API password
- `apiKey=newKey`: string - Changes the API Key (set to an empty string do disable it)
- `custom_headers={[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".

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 an 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.

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 accepted, over UDP, TCP and DNS over TLS. The default is empty. Note 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.

setProxyProtocolApplyACL (*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

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*)
New in version 1.3.0.

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.

setRingBuffersSize (*num*[, *numberOfShards*])
Changed in version 1.3.0: `numberOfShards` optional parameter added.

Set the capacity of the ringbuffers used for live traffic inspection to `num`, and the number of shards to `numberOfShards` if specified.

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. Defaults to 1

17.1.3 Servers

newServer (*server_string*)
newServer (*server_table*)
Changed in version 1.3.0: Added `checkClass`, `sockets` and `checkFunction` to 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.

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({
  address="IP:PORT",      -- IP and PORT of the backend server (mandatory)
  id=STRING,              -- Use a pre-defined UUID instead of a random one
  qps=NUM,                -- Limit the number of queries per second to NUM,␣
↪when using the `firstAvailable` policy
```

(continues on next page)

(continued from previous page)

```
  order=NUM,              -- The order of this server, used by the␣
↪`leastOutstanding` and `firstAvailable` policies
  weight=NUM,             -- 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=NUM,            -- The number of TCP connection attempts to the␣
↪backend, for a given query
  tcpConnectTimeout=NUM, -- The timeout (in seconds) of a TCP connection␣
↪attempt
  tcpSendTimeout=NUM,     -- The timeout (in seconds) of a TCP write attempt
  tcpRecvTimeout=NUM,     -- 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=NUM,         -- Use NUM 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."
  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 :ref:`Healthcheck`)
  checkTimeout=NUM,       -- 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=NUM,   -- Allow NUM check failures before declaring the␣
↪backend down, default: 1
  checkInterval=NUM       -- 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 this backend, by default this is left to the kernel's address selection
                          -- 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=NUM,             -- 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 <https://datatracker.ietf.org/doc/draft-bellis-dnsop-xpf/>`_
↪ and the specified option code. Default is disabled (0)
  sockets=NUM,            -- Number of sockets (and thus source ports) used␣
↪toward the backend server, defaults to a single one
  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=NUM,               -- Require NUM 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.
```

(continues on next page)

(continued from previous page)

```
  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             -- 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 :func:`addLocal`␣
↪and/or :func:`addTLSLocal`. Note that out-of-order is always enabled on DoH␣
↪frontends.
})
```

Parameters

- **server_string** (*str*) – A simple IP:PORT string.
- **server_table** (*table*) – A table with at least a 'name' key

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*])
Changed in version 1.3.0: `status` optional parameter added.

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

:setQPS(*limit*)
Limit the queries per second for this server.

Parameters **limit** (*int*) – The maximum number of queries per second

:setDown()
Set the server in an `DOWN` state. The server will not receive queries and the health checks are disabled

: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

17.1.4 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 *addPool()*.

addPool(*name*) → ServerPool
Returns a *ServerPool*.

Parameters **name** (*string*) – The name of the pool to create

getPool(*name*) → ServerPool
Returns a *ServerPool* or nil.

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

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()
New in version 1.3.0.

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()
New in version 1.3.0.

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
Changed in version 1.3.0: `numberOfShards` and `deferrableInsertLock` parameters added.

Changed in version 1.3.1: `maxNegativeTTL` and `parseECS` parameters added.

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.

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=1`: int - Number of shards to divide the cache into, to reduce lock contention.
- `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 - Whether 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.

class PacketCache
Represents a cache that can be part of *ServerPool*.

:dump (*fname*)
New in version 1.3.1.

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 1.2.0: `suffixMatch` parameter added.

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`

: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

17.1.5 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.

: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.

17.1.6 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.

getTLSContext(*idx*)
New in version 1.3.0.

Return the TLSContext object for the context of index `idx`.

getTLSFrontend(*idx*)
New in version 1.3.1.

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.

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.

getTopRules([*top*])
New in version 1.6.0.

Return the rules that matched the most.

Parameters **top** (*int*) – How many rules to return.

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.

grepq(*selector*[, *num*])
grepq(*selectors*[, *num*])
Prints the last `num` queries matching `selector` or `selectors`.

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, default is 10.

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

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 availables DNS over HTTPS frontends.

showDOHResponseCodes()
New in version 1.4.0.

Print the HTTP response codes statistics for all availables DNS over HTTPS 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``()`
New in version 1.3.0.

Print the list of all availables 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.

`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

17.1.7 Dynamic Blocks

addDynBlocks (*addresses*, *message*[, *seconds=10*[, *action*]])
Changed in version 1.2.0: `action` parameter added.

Block a set of addresses with `message` for (optionally) a number of seconds. The default number of seconds to block for is 10.

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 *here*. (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.

showDynBlocks ()
List all dynamic blocks in effect.

setDynBlocksAction (*action*)
Changed in version 1.3.3: `DNSAction.NXDomain` action added.

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 300 (5 minutes), 0 means disabled.

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
New in version 1.3.0.

Creates a new *DynBlockRulesGroup* object.

class DynBlockRulesGroup
Represents a group of dynamic block rules.

:setQueryRate (*rate*, *seconds*, *reason*, *blockingTime*[, *action*[, *warningRate*]])
Changed in version 1.3.3: `warningRate` parameter added.

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 *here*. (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

:setRCodeRate (*rcode*, *rate*, *seconds*, *reason*, *blockingTime*[, *action*[, *warningRate*]])
Changed in version 1.3.3: `warningRate` parameter added.

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 *here*. (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.

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** (*int*) – 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 *here*. (default to the one set with *setDynBlocksAction()*)
- **warningRatio** (*int*) – 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*]])

Changed in version 1.3.3: `warningRate` parameter added.

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 *here*. (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*]])
Changed in version 1.3.3: `warningRate` parameter added.

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 *here*. (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.

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 second one with the stats of the current label and one 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 label 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.

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 *here*. (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 label 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 *here*. (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*)
New in version 1.3.1.

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 netmask, or list of netmasks, as strings, like for example "192.0.2.1/24"

:includeRange(*netmasks*)
New in version 1.3.1.

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 netmask, or list of netmasks, as strings, like for example "192.0.2.1/24"

:toString()
New in version 1.3.1.

Return a string describing the rules and range exclusions of this DynBlockRulesGroup.

StatNode

class StatNode
Represent metrics about a given node, for the visitor functions used with *DynBlockRulesGroup:setSuffixMatchRule()* and *DynBlockRulesGroup:setSuffixMatchRuleFFI()*. Note that some nodes includes the metrics for their children as well as their own.

bytes
The number of bytes for all responses returned for that node.

drops
The number of drops for that node.

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'.

noerrors
The number of No Error answers returned 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.

:numChildren()
The number of children of 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.

: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.

: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.

17.1.8 Other functions

maintenance()
If this function exists, it is called every second to do regular tasks. This can be used for e.g. *Dynamic Blocks*.

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.

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)

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.

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.

: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.

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()*.

TLSContext

class TLSContext
New in version 1.3.0.

This object represents an address and port dnsdist is listening on for DNS over TLS queries.

:rotateTicketsKey()
Replace the current TLS tickets key by a new random one.

: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.

Parameters **ticketsKeysFile** (*str*) – The path to a file from where TLS tickets keys should be loaded.

TLSFrontend

class TLSFrontend
New in version 1.3.1.

This object represents the configuration of a listening frontend for DNS over TLS queries. To each frontend is associated a TLSContext.

TLSContext:**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.

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*)
New in version 1.3.3.

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*)
New in version 1.3.3.

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*)
New in version 1.3.3.

Set the interval, in seconds, between two security pollings.

Parameters **interval** (*int*) – The interval, in seconds, between two pollings. Default is 3600.

setSecurityPollSuffix (*suffix*)
New in version 1.3.3.

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.'.

17.2 Constants

There are many constants in **dnsdist**.

17.2.1 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

17.2.2 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

17.2.3 RCode

These constants represent the different RCODEs for DNS messages.

Changed in version 1.4.0: The prefix is changed from `dnsdist` to `DNSRCode`.

- `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

17.2.4 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

17.2.5 DNS Packet Sections

These constants represent the section in the DNS Packet.

- DNSSection.Question
- DNSSection.Answer
- DNSSection.Authority
- DNSSection.Additional

17.2.6 DNSAction

Changed in version 1.5.0: `DNSAction.SpoofRaw` 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.SpoofRaw`: spoof the response using the supplied raw value as record data
- `DNSAction.Truncate`: truncate the response
- `DNSAction.NoRecurse`: set rd=0 on the query

17.2.7 DNSQType

Changed in version 1.4.0: The prefix is changed from `dnsdist.` to `DNSQType`.

All named QTypes are available as constants, prefixed with `DNSQType.`, e.g.:

- `DNSQType.AAAA`
- `DNSQType.AXFR`
- `DNSQType.A`
- `DNSQType.NS`
- `DNSQType.SOA`
- etc.

17.2.8 DNSResponseAction

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

17.3 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

17.4 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`.

17.5 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 values are integers, as returned by *exceed** functions.

: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.

17.6 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
```

17.6.1 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
New in version 1.2.0.

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

: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.

:wirelength() → int
Returns the length in bytes of the DNSName as it would be on the wire.

17.7 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"))
```

17.7.1 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.

17.8 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:

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*.

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.

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.

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

:getDO() → bool
New in version 1.2.0.

Get the value of the DNSSEC OK bit.

Returns true if the DO bit was set, false otherwise

:getEDNSOptions() → table
New in version 1.3.3.

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.

Return the HTTP headers for a DoH query, as a table whose keys are the header names and values the header values.

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`

: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

: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
New in version 1.2.0.

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
New in version 1.2.0.

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

:sendTrap(*reason*)
New in version 1.2.0.

Send an SNMP trap.

Parameters **reason** (*string*) – An optional string describing the reason why this trap was sent

: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*)

:setTag(*key*, *value*)
New in version 1.2.0.

Set a tag into the DNSQuestion object.

Parameters

- **key** (*string*) – The tag's key
- **value** (*string*) – The tag's value

:setTagArray(*tags*)
New in version 1.2.0.

Set an array of tags into the DNSQuestion object.

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

17.9 DNSResponse object

class DNSResponse
This object has all the functions and members of a *DNSQuestion* and some more

: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.

17.10 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.

:getRA() → bool
Get recursion available flag.

:getRD() → bool
Get recursion desired 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

17.11 EDNSOptionView object

class EDNSOptionView
New in version 1.3.3.

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.

17.12 eBPF functions and objects

These are all the functions, objects and methods related to the *eBPF Socket Filtering*.

addBPFFilterDynBlocks(*addresses*, *dynbpf*[[, *seconds=10*], *msg*])
Changed in version 1.3.0: `msg` optional parameter added.

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.

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(*maxV4*, *maxV6*, *maxQNames*) → BPFFilter
Return a new eBPF socket filter with a maximum of maxV4 IPv4, maxV6 IPv6 and maxQNames qname entries in the block table.

Parameters

- **maxV4** (*int*) – Maximum number of IPv4 entries in this filter
- **maxV6** (*int*) – Maximum number of IPv6 entries in this filter
- **maxQNames** (*int*) – Maximum number of QName entries in this filter

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()*

:block(*address*)
Block this address

Parameters **address** (ComboAddress) – The address to block

: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

: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.

:purgeExpired()
Remove the expired ephemeral rules associated with this filter.

:excludeRange(*netmasks*)
New in version 1.3.3.

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 **netmasks** (*int*) – A netmask, or list of netmasks, as strings, like for example "192.0.2.1/24"

:includeRange(*netmasks*)
New in version 1.3.3.

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 **netmasks** (*int*) – A netmask, or list of netmasks, as strings, like for example "192.0.2.1/24"

17.13 DNSCrypt objects and functions

addDNSCryptBind(*address*, *provider*, *certFile(s)*, *keyFile(s)*[, *options*])
Changed in version 1.3.0: `cpus` option added.

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.

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.

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*])
Changed in version 1.3.0: `version` optional parameter added.

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.

17.13.1 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

17.13.2 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.

17.13.3 Context

class DNSCryptContext
Represents a DNSCrypt content. Can be used to rotate certs.

:addNewCertificate(*cert*, *key*[, *active*])
New in version 1.3.0.

Add a new certificate to the 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*])
Changed in version 1.3.0: `version` optional parameter added.

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)

:getCurrentCertificate() → DNSCryptCert

Deprecated since version 1.3.0: Removed as it relied on one certificate. See *DNSCryptContext:getCertificate()*.

Return the current certificate.

:getOldCertificate() → DNSCryptCert

Deprecated since version 1.3.0: Removed as it relied on one certificate.

Return the previous certificate.

:hasOldCertificate() → bool
Deprecated since version 1.3.0: Removed as it relied on one certificate.

Whether or not the context has a previous certificate, from a certificate rotation.

:getCertificate(*index*) → DNSCryptCert
New in version 1.3.0.

Return the certificate with index *index*.

Parameters **index** (*int*) – The index of the certificate, starting at 0

:getCertificatePair(*index*) → DNSCryptCertificatePair
New in version 1.3.0.

Return the certificate pair with index *index*.

Parameters **index** (*int*) – The index of the certificate, starting at 0

:getCertificatePair(*index*) → table of DNSCryptCertificatePair
New in version 1.3.0.

Return a table of certificate pairs.

:getProviderName() → string
Return the provider name

:loadNewCertificate(*certificate*, *keyfile*[, *active*])
Changed in version 1.3.0: `active` optional parameter added.

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*)
New in version 1.3.0.

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*)
New in version 1.3.0.

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()
New in version 1.3.0.

Print all the certificates.

:removeInactiveCertificate(*serial*)
New in version 1.3.0.

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

17.14 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*)
New in version 1.2.0.

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*)
New in version 1.2.0.

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*)
New in version 1.3.3.

Set the server identify field.

Parameters **id** (*string*) – The server ID

:setTag(*value*)
New in version 1.2.0.

Add a tag to the list of tags.

Parameters **value** (*string*) – The tag value

:setTagArray(*valueList*)
New in version 1.2.0.

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

17.15 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

17.16 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

17.17 SNMP reporting

New in version 1.2.0.

snmpAgent(*enableTraps*[, *masterSocket*])
Enable SNMP support.

Parameters

- **enableTraps** (*bool*) – Indicates whether traps should be sent
- **masterSocket** (*string*) – A string specifying how to connect to the master 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

17.18 Tuning related functions

setMaxTCPClientThreads(*num*)
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.

Parameters **num** (*int*) –

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*)
Set the maximum number of TCP connections queued (waiting to be picked up by a client thread), defaults to 1000. 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

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*) –

setStaleCacheEntriesTTL(*num*)
Allows using cache entries expired for at most n seconds when no backend available to answer for a query

Parameters **num** (*int*) –

setTCPUseSinglePipe(*val*)
Whether the incoming TCP connections should be put into a single queue instead of using per-thread queues. Defaults to false

Parameters **val** (*bool*) –

setTCPRecvTimeout(*num*)
Set the read timeout on TCP connections from the client, in seconds

Parameters **num** (*int*) –

setTCPSendTimeout(*num*)
Set the write timeout on TCP connections from the client, in seconds

Parameters **num** (*int*) –

setUDPMultipleMessagesVectorSize(*num*)
New in version 1.3.0.

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

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*) –

17.19 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 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.

```
> 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.

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)

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() → 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.

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 'refrehDelay' 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*) → KeyValueStore
New in version 1.4.0.

Return a new KeyValueStore object associated to the corresponding LMDB database. The database must have been created with the `MDB_NOSUBDIR` flag.

Parameters

- **filename** (*string*) – The path to an existing LMDB database created with `MDB_NOSUBDIR`
- **dbName** (*string*) – The name of the database to use

17.20 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.

17.21 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.

CHAPTER

EIGHTEEN

MANUAL PAGES

18.1 dnsdist

18.1.1 Synopsis

dnsdist [<option>...] [address]...

18.1.2 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.

18.1.3 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.

18.1.4 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 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).

-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.

18.1.5 Bugs

Right now, the TCP support has some rather arbitrary limits.

18.1.6 Resources

Website: https://dnsdist.org

CHAPTER

NINETEEN

CHANGELOG

19.1 1.5.1

Released: 1st of October 2020

19.1.1 Improvements

- Add the 'clearConsoleHistory' command ¶ References: #9372, pull request 9540

19.1.2 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

19.2 1.5.0

Released: 30th of July 2020

19.2.1 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

19.2.2 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

19.3 1.5.0-rc4

Released: 7th of July 2020

19.3.1 Bug Fixes

- Prevent a race between the DoH handling threads ¶ References: pull request 9278

19.4 1.5.0-rc3

Released: 18th of June 2020

19.4.1 New Features

- Implement an ACL in the internal web server ¶ References: pull request 9229

19.4.2 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

19.4.3 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

19.5 1.5.0-rc2

Released: 13th of May 2020

19.5.1 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

19.5.2 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

19.6 1.5.0-rc1

Released: 16th of April 2020

19.6.1 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

19.6.2 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

19.7 1.5.0-alpha1

Released: 20th of March 2020

19.7.1 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
- Dnsdist: 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

19.7.2 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
- Document that the 'keyLogFile' option requires OpenSSL >= 1.1.1 ¶ References: #8806, pull request 8899
- 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
- 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

19.7.3 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

19.8 1.4.0

Released: 20th of November 2019

19.8.1 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

19.8.2 Bug Fixes

- Lowercase the name blocked by a SMT dynamic block ¶ References: pull request 8524

19.8.3 misc

- Prefer the cipher suite from the server by default (DoH, DoT) ¶ References: pull request 8526

19.9 1.4.0-rc5

Released: 30th of October 2019

19.9.1 Improvements

- Rename the 'address' label to 'frontend' for DoH metrics ¶ References: pull request 8465

19.9.2 Bug Fixes

- Increment the DOHUnit ref count when it's set in the IDState ¶ References: pull request 8471

19.10 1.4.0-rc4

Released: 25th of October 2019

19.10.1 New Features

- Add support dumping TLS keys via keyLogFile ¶ References: pull request 8442

19.10.2 Improvements

- Implement reference counting for the DOHUnit object ¶ References: pull request 8416
- 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
- Count the number of concurrent connections for DoH as well ¶ References: pull request 8395
- Add a 'preferServerCiphers' option for DoH and DoT ¶ References: pull request 8382
- Lowercase custom DoH header names ¶ References: #8353, pull request 8365
- Refactor DoH prometheus metrics again ¶ References: pull request 8361
- Add metrics about TLS versions with DNS over TLS ¶ References: pull request 8387
- 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

19.10.3 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

19.11 1.4.0-rc3

Released: 30th of September 2019

19.11.1 Improvements

- Display the DoH and DoT binds in the web view ¶ References: pull request 8264
- 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
- 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

19.11.2 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

19.12 1.4.0-rc2

Released: 2nd of September 2019

19.12.1 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

19.12.2 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

19.12.3 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

19.13 1.4.0-rc1

Released: 12th of August 2019

19.13.1 New Features

- Add OCSP stapling (from files) for DoT and DoH ¶ References: #7812, pull request 8141
- Add support for custom DoH headers (Melissa Voegeli) ¶ References: #7957, #7900, pull request 8148
- Add lua bindings, rules and action for DoH ¶ References: #8133, pull request 8153
- Implement ContinueAction() ¶ References: pull request 8117

19.13.2 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

19.13.3 Bug Fixes

- Properly override the HTTP Server header for DoH ¶ References: #7894, pull request 7911
- Exit when requested DoT/DoH support is not compiled in ¶ References: pull request 7915
- Proper HTTP response for timeouts over DoH ¶ References: #7917, pull request 7927
- Prevent a dangling DOHUnit pointer when send() failed ¶ References: pull request 8112
- 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: #8084, #6942, 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

19.14 1.4.0-beta1

Released: 6th of June 2019

19.14.1 New Features

- Implement SNIRule for DoT and DoH ¶ References: #7210, pull request 7825

19.14.2 Improvements

- Support Prometheus latency histograms (Marlin Cremers) ¶ References: #6088, pull request 7853

19.14.3 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

19.15 1.4.0-alpha2

Released: 26th of April 2019

19.15.1 New Features

- Add DNS over HTTPS support based on libh2o ¶ References: #7526, #6911, pull request 7726

19.15.2 Improvements

- Ignore Path MTU discovery on UDP server socket ¶ References: pull request 7410
- Alternative solution to the unaligned accesses. ¶ References: pull request 7708

19.15.3 Bug Fixes

- Exit when setting ciphers fails (GnuTLS) ¶ References: pull request 7718

19.16 1.4.0-alpha1

Released: 12th of April 2019

19.16.1 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: #6897, #6846, 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

19.16.2 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: #7526, #4814, 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

19.16.3 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

19.17 1.3.3

Released: 8th of November 2018

19.17.1 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 7007, pull request 7089, pull request 6901, pull request 3935, pull request 6343
- 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

19.17.2 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: #6348, #4857, 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: #7004, #6990, 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

19.17.3 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

19.18 1.3.2

Released: 10th of July 2018

19.18.1 Bug Fixes

- Add missing include for PRId64, fix build on CentOS 6 / SLES 12 ¶ References: pull request 6785

19.19 1.3.1

Released: 10th of July 2018

19.19.1 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

19.19.2 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 6470, pull request 6445, pull request 6457
- 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: #6709, #6683, 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

19.19.3 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

19.20 1.3.0

Released: 30th of March 2018

19.20.1 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 6177, pull request 6189, pull request 6176, pull request 6175
- 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 6220, pull request 5594
- 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

19.20.2 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 6366, pull request 6350
- 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 6375, pull request 5866
- 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

19.20.3 Bug Fixes

- Handle SNMP alarms so we can reconnect to the master. ¶ 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

19.20.4 Removals

- Remove the `--daemon` option from **dnsdist**. ¶ References: #6329, pull request 6394

19.21 1.2.1

Released: 16th of February 2018

19.21.1 New Features

- Add configuration option to disable IP_BIND_ADDRESS_NO_PORT (Dan McCombs). ¶ References: pull request 5880

19.21.2 Improvements

- Handle bracketed IPv6 addresses without ports (Chris Hofstaedtler). ¶ References: pull request 6057

19.21.3 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

19.22 1.2.0

Released: 21st of August 2017

19.22.1 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 5036, pull request 4788
- 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 5508, pull request 5490
- Add the capability to set arbitrary tags in protobuf messages. ¶ References: pull request 5577, pull request 5396
- Add setConsoleConnectionsLogging(). ¶ References: #5565, pull request 5581

19.22.2 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 5501, pull request 4985
- 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: #4155, #4131, 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: #4775, #4660, 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

19.22.3 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

19.22.4 Removals

- Remove BlockFilter. ¶ References: #5513, pull request 5514
- Deprecate syntactic sugar functions. ¶ References: #5069, pull request 5526

19.22.5 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

19.23 1.1.0

Released December 29th 2016

Changes since 1.1.0-beta2:

19.23.1 Improvements

- #4783: Add -latomic on powerpc
- #4812: Handle header-only responses, handle Refused as Servfail in the cache

19.23.2 Bug fixes

- #4762: SuffixMatchNode: Fix an insertion issue for an existing node
- #4772: Fix dnsdist initscript config check

19.24 1.1.0-beta2

Released December 14th 2016

Changes since 1.1.0-beta1:

19.24.1 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()`

19.24.2 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

19.24.3 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

19.25 1.1.0-beta1

Released September 1st 2016

Changes since 1.0.0:

19.25.1 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)

19.25.2 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

19.25.3 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

19.26 1.0.0

Released April 21st 2016

Changes since 1.0.0-beta1:

19.26.1 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

19.26.2 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

19.27 1.0.0-beta1

Released April 14th 2016

Changes since 1.0.0-alpha2:

19.27.1 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

19.27.2 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

19.27.3 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

19.28 1.0.0-alpha2

Released February 5th 2016

Changes since 1.0.0-alpha1:

19.28.1 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)

19.28.2 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 crashs (#3277)
- Useless sockets were created in client mode (#3257)
- `addAnyTCRule()` was generating TC=1 responses even over TCP (#3251)

19.28.3 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

19.28.4 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

19.29 1.0.0-alpha1

Released December 24th 2015

Initial release

CHAPTER

TWENTY

UPGRADE GUIDE

20.1 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.

20.2 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".

20.3 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`

20.4 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.

20.5 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()*

20.6 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 `addRule()` combined with a *DropAction()* can do the same.

CHAPTER

TWENTYONE

SECURITY ADVISORIES

All security advisories for the DNSDist are listed here.

21.1 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.

21.2 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

21.3 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.

CHAPTER

TWENTYTWO

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

HTTP ROUTING TABLE

/api

/jsonstat

/metrics

INDEX

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W