NSD User Manual

Release 4.3.9

NLnet Labs

Apr 24, 2024

GETTING STARTED

Welcome to the documentation of the NLnet Labs Name Server Daemon (NSD), an authoritative DNS name server. It has been developed for operations in environments where speed, reliability, stability and security are of high importance.

NSD is designed with a pure philosophy that prioritises raw performance. This means that if you serve hundreds of thousands or even millions of queries per second, NSD is the leading implementation in the world. This authoritative DNS name server strives to be a reference implementation for emerging standards of the Internet Engineering Task Force (IETF). NSD is distributed free of charge in open source form under the BSD license. For most platforms, packages are available.

This documentation is an open source project maintained by NLnet Labs. is edited via text files in the reStructuredText markup language and then compiled into a static website/offline document using the open source Sphinx and ReadTheDocs tools.

We always appreciate your feedback and improvements. You can submit an issue or pull request on the GitHub repository, or post a message on the NSD users mailing list. All the contents are under the permissive Creative Commons Attribution 3.0 (CC-BY 3.0) license, with attribution to NLnet Labs.

CHAPTER ONE

INSTALLATION

To install your own copy of NSD you have two options: use the version provided by your package manager, or download the source and building it yourself.

Installing via the package manager is the easiest option, and on most systems even trivial. The downside is the distributed version can be outdated for some distributions or not have all the compile-time options included that you want. Building and compiling NSD yourself ensures that you have the latest version and all the compile-time options you desire.

1.1 Introduction

NSD consists of two programs: the zone compiler `zonec` and the name server `nsd` itself. The name server works with an intermediate database prepared by the zone compiler from standard zone files.

For NSD operation this means that zones have to be compiled by `zonec` before NSD can use them. All this can be controlled via `rc.d` (SIGTERM, SIGHUP) or **`nsd-control`**, and uses a simple configuration file `nsd.conf`.

1.2 Installing with a package manager

Most package managers maintain a version of NSD, although this version can be outdated if this package has not been updated recently. If you like to upgrade to the latest version, we recommend compiling NSD yourself.

1.2.1 Debian/Ubuntu

Installing NSD with the built-in package manager should be as easy as:

```
sudo apt update
sudo apt install nsd
```

This gives you a compiled and running version of NSD ready to *be configured*.

1.3 Building from source/Compiling

1.3.1 Ubuntu 20.04 LTS

First of all, we need our copy of the NSD code. On our website you can find the latest version and the changelog. In this example we'll use version 4.3.7. Please note that this is not the latest version currently.

```
wget https://nlnetlabs.nl/downloads/nsd/nsd-4.3.7.tar.gz
tar xzf nsd-4.3.7.tar.gz
```

We'll need some tools, such as a compiler and the **make** program.

```
sudo apt update
sudo apt install -y build-essential
```

The library components NSD needs are: `libssl` and `libevent`, of which we need the "dev" version.

```
sudo apt install -y libssl-dev
sudo apt install -y libevent-dev
```

We'll also need the tools to build the actual program. For this, NSD uses **make** and internally it uses `flex` and `yacc`, which we need to download as well.

```
sudo apt install -y bison
sudo apt install -y flex
```

With all the requirements met, we can now start the compilation process in the NSD directory. The first step here is configuring. With `./configure -h` you can look at the extensive list of configurables for NSD. A nice feature is that **configure** will tell you what it's missing during configuration.

```
./configure
```

If **configure** gives no errors, we can continue to actually try compiling NSD using **make**; compilation might take a while.

```
make
```

After successfully compiling, we can install NSD to make it available for the machine.

```
sudo make install
```

We now have fully compiled and installed version of NSD, and can continue to testing it.

1.4 Testing

A simple test to determine if the installation was successful is to invoke the **nsd** command with the *-V* option, which is the "version" option. This shows the version and build options used and proves installation was successful.

```
nsd -v
```

If all the previous steps were successful we can continue to configuring our NSD instance.

Another handy trick you can use during testing is to run NSD in the foreground using the *-d* option and increase the verbosity level using the `-V 3` option. This allows you to see steps NSD takes and also where it fails.

Now that NSD is installed we can *continue to configuring it*.

CHAPTER

TWO

CONFIGURATION

NSD has a vast array of configuration options for advanced use cases. To configure the application, a `nsd.conf` configuration file used. The file format has attributes and values, and some attributes have attributes inside them.

Note: The instructions in this page assume that NSD is already installed.

2.1 The configuration file

The configuration NSD uses is specified in the configuration file, which can be supplied to NSD using the `-c` option. In the *reference* an example `nsd.conf` can be found as well as the complete documentation of all the configurable options. The same example and reference can be found on your system using the `man nsd.conf` command.

The basic rules are of the config file are:

- The used notation is `attribute:  value`
- Comments start with # and extend to the end of a line
- Empty lines are ignored, as is whitespace at the beginning of a line
- Quotes can be used, for names containing spaces, e.g. `"file name.zone"`

Below we'll give an example config file, which specifies options for the NSD server, zone files, primaries and secondaries. This provide basic config which can be used for as a starting point.

Note that for the remainder we assume the default location of NSD is `/etc/nsd` though this may vary on your system.

The example configuration below specifies options for the NSD server, zone files, primaries and secondaries.

Here is an example config for `example.com`:

```
server:
    # use this number of cpu cores
    server-count: 1
    # We recommend leaving this empty, otherwise use "/var/db/nsd/nsd.db"
    database: ""
    #  the default file used for the nsd-control addzone and delzone commands
    # zonelistfile: "/var/db/nsd/zone.list"
    # The unprivileged user that will run NSD, can also be set to "" if
    # user privilige protection is not needed
    username: nsd
    # Default file where all the log messages go
    logfile: "/var/log/nsd.log"
```

(continues on next page)

(continued from previous page)

```
    # Use this pid file instead of the platform specific default
    pidfile: "/var/run/nsd.pid"
    # Enable if privilege "jail" is needed for unprivileged user. Note
    # that other file paths may break when using chroot
    # chroot: "/etc/nsd/"
    # The default zone transfer file
    # xfrdfile: "/var/db/nsd/xfrd.state"
    # The default working directory before accessing zone files
    # zonesdir: "/etc/nsd"

remote-control:
    # this allows the use of 'nsd-control' to control NSD. The default is "no"
    control-enable: yes
    # the interface NSD listens to for nsd-control. The default is 127.0.0.1
    control-interface: 127.0.0.1
    # the key files that allow the use of 'nsd-control'. The default path is "/etc/nsd/".␣
↪Create these using the 'nsd-control-setup' utility
    server-key-file: /etc/nsd/nsd_server.key
    server-cert-file: /etc/nsd/nsd_server.pem
    control-key-file: /etc/nsd/nsd_control.key
    control-cert-file: /etc/nsd/nsd_control.pem

zone:
    name: example.com
    zonefile: /etc/nsd/example.com.zone
```

We recommend not using the database (so using `database: ""`) as this is will slow down NSD operation. Depending on your needs, we also recommend keeping the `server-count` lower or equal to the number of CPU cores your system has.

Optionally, you can control NSD (from the same or even a different device) by using the entries under the *remote-control* clause in the config. Using this tool, NSD can be controlled (find the reference of all the options *here*) which makes controlling NSD much easier. If your install does not come with the keys needed for remote-control use pre-made, you can generate the keys using the **`nsd-control-setup`** command, which will create them for you. In the section below we will go into more detail about this option.

You can test the config with **`nsd-checkconf`**. This tool will tell you what is wrong with the config and where the error occurs.

If you are happy with the config and any modifications you may have done, you can create the zone to go with the file we mentioned in the config. We show an example zone at *the zonefile example*.

2.2 Setting up a secondary zone

If your needs go further than just a few zones that are managed locally, NSD has got you covered. We won't go into the theoretical details of primaries and secondaries here (we recommend this blog), but we will show how to configure it.

The example for a secondary looks like this:

```
zone:
    # this server is the primary, 192.0.2.1 is the secondary.
    name: primaryzone.com
```

(continues on next page)

(continued from previous page)

```
    zonefile: /etc/nsd/primaryone.com.zone
    notify: 192.0.2.1 NOKEY # NOKEY for testing purposes only
    provide-xfr: 192.0.2.1 NOKEY # NOKEY for testing purposes only

zone:
    # this server is secondary, 192.0.2.2 is primary.
    name: secondaryzone.com
    zonefile: /etc/nsd/secondaryzone.com.zone
    allow-notify: 192.0.2.2 NOKEY # NOKEY for testing purposes only
    request-xfr: 192.0.2.2 NOKEY # NOKEY for testing purposes only
```

Note: Note that the NOKEY keyword above are for testing purposes only, as this can introduce vulnerabilities when used in production environments.

For a secondary zone we list the primaries by IP address. Below is an example of a secondary zone with two primary servers. If a primary only supports AXFR transfers and not IXFR transfers (like NSD), specify the primary as `request-xfr:  AXFR <ip_address> <key>`. By default, all zone transfer requests are made over TCP. If you want the IXFR request be transmitted over UDP, use `request-xfr:  UDP <ip address> <key>`.

```
zone:
  name: "example.com"
  zonefile: "example.com.zone"
  allow-notify: 168.192.185.33 NOKEY
  request-xfr: 168.192.185.33 NOKEY
  allow-notify: 168.192.199.2 NOKEY
  request-xfr: 168.192.199.2 NOKEY
```

By default, a secondary will fallback to AXFR requests if the primary told us it does not support IXFR. You can configure the secondary not to do AXFR fallback with:

```
allow-axfr-fallback: "no"
```

For a primary zone, list the secondary servers, by IP address or subnet. Below is an example of a primary zone with two secondary servers:

```
zone:
    name: "example.com"
    zonefile: "example.com.zone"
    notify: 168.192.133.75 NOKEY
    provide-xfr: 168.192.133.75 NOKEY
    notify: 168.192.5.44 NOKEY
    provide-xfr: 168.192.5.44 NOKEY
```

You also can set the outgoing interface for notifies and zone transfer requests to satisfy access control lists at the other end:

```
outgoing-interface: 168.192.5.69
```

By default, NSD will retry a notify up to five times. You can override that value with:

```
notify-retry: 5
```

Zone transfers can be secured with TSIG keys, replace NOKEY with the name of the TSIG key to use. See *Using TSIG* for details.

Since NSD is written to be run on the root name servers, the config file can to contain something like:

```
zone:
    name: "."
    zonefile: "root.zone"
    provide-xfr: 0.0.0.0/0 NOKEY # allow axfr for everyone.
    provide-xfr: ::0/0 NOKEY
```

You should only do that if you're intending to run a root server, NSD is not suited for running a `.` cache. Therefore if you choose to serve the `.` zone you have to make sure that the complete root zone is timely and fully updated.

To prevent misconfiguration, NSD configure has the `--enable-root-server` option, that is by default disabled.

In the config file, you can use patterns. A pattern can have the same configuration statements that a zone can have. And then you can `include-pattern: <name-of-pattern>` in a zone (or in another pattern) to apply those settings. This can be used to organise the settings.

2.3 Remote controlling NSD

The **`nsd-control`** tool is also controlled from the `nsd.conf` config file (and it's manpage is found *here*). It uses TLS encrypted transport to 127.0.0.1, and if you want to use it you have to setup the keys and also edit the config file. You can leave the remote-control disabled (the secure default), or opt to turn it on:

```
# generate keys
nsd-control-setup
```

```
# edit nsd.conf to add this
remote-control:
  control-enable: yes
```

By default **`nsd-control`** is limited to localhost, as well as encrypted, but some people may want to remotely administer their nameserver. To control NSD remotely, configure **`nsd-control`** to listen to the public IP address with `control-interface: <IP>` after the control-enable statement.

Furthermore, you copy the key files `/etc/nsd/nsd_server.pem` `/etc/nsd/nsd_control.*` to a remote host on the internet; on that host you can run **`nsd-control`** with `-c` `<special config file>` which references same IP address `control-interface` and references the copies of the key files with `server-cert-file`, `control-key-file` and `control-cert-file` config lines after the `control-enable` statement. The nsd-server authenticates the nsd-control client, and also the **`nsd-control`** client authenticates the nsd-server.

2.4 Starting up the first time

When you are done with the configuration file, check the syntax using

```
nsd-checkconf <name of configfile>
```

The zone files are read by the daemon, which builds `nsd.db` with their contents. You can start the daemon in a number of ways:

```
nsd -c <name of configfile>
nsd-control start # which execs nsd via the remote-control configuration
nsd # which will use the default configuration file
```

To check if the daemon is running look with **`ps`**, **`top`**, or if you enabled **`nsd-control`**:

```
nsd-control status
```

To reload changed zone files after you edited them, without stopping the daemon, use this to check if files are modified:

```
kill -HUP `cat <name of nsd pidfile>`
or "nsd-control reload" if you have remote-control enabled
```

With **`nsd-control`** you can also reread the config file, in case of new zones, etc.

```
nsd-control reconfig
```

To restart the daemon:

```
/etc/rc.d/nsd restart    # or your system(d) equivalent
```

To shut it down (for example on the system shutdown) do:

```
kill -TERM <pid of nsd>
or nsd-control stop
```

NSD will automatically keep track of secondary zones and update them when needed. When primary zones are updated and reloaded notifications are sent to secondary servers.

The zone transfers are applied to `nsd.db` by the daemon. To write changed contents of the zone files for secondary zones to disk in the text-based zone file format, issue **`nsd-control`** write.

NSD will send notifications to secondary zones if a primary zone is updated. NSD will check for updates at primary servers periodically and transfer the updated zone by AXFR/IXFR and reload the new zone contents.

If you wish exert manual control use **`nsd-control notify`**, **`transfer`** and **`force_transfer`** commands. The transfer command will check for new versions of the secondary zones hosted by this NSD. The notify command will send notifications to the secondary servers configured in `notify:` statements.

CHAPTER

THREE

ZONEFILE EXAMPLE

On this page we give an example of a basic zone file and it's contents.

We recommend using the **nsd-checkzone** tool to verify that you have a working zone.

3.1 Creating a zone

A minimal zone needs exactly one SOA (Source Of Authority) and one or more NS (Name Server) records. Refer to appropriate documentation of you need to learn about DNS basics.

```
$ORIGIN example.com.
$TTL 86400 ; default time-to-live for this zone

example.com.   IN  SOA     ns.example.com. noc.dns.example.org. (
        2020080302  ;Serial
        7200        ;Refresh
        3600        ;Retry
        1209600     ;Expire
        3600        ;Negative response caching TTL
)

; The nameservers that are authoritative for this zone.
                                NS      example.com.

; A and AAAA records are for IPv4 and IPv6 addresses respectively
example.com.    A       192.0.2.1
                                AAAA 2001:db8::3

; A CNAME redirects from www.example.com to example.com
www                             CNAME   example.com.

mail                    MX      10      example.com.
```

CHAPTER

FOUR

CATALOG ZONES

Since version 4.9.0, NSD has support for Catalog zones version "2" as specified in RFC 9432. NSD can be a producer of catalog zones as well as a catalog zone consumer, but it is limited to process only a single consumer zone.

4.1 Setting up NSD as a catalog consumer

NSD will process a zone as a catalog consumer zone if the zone has the `catalog: consumer` option set. An example catalog consumer configuration could look like this:

```
pattern:
        name: "member-zone-config"
        request-xfr: 198.51.100.1 NOKEY
        allow-notify: 198.51.100.1 NOKEY

key:
        name: tsig-key.name
        algorithm: hmac-sha256
        secret: "SXMgdGhpcyBhIHNlY3JldCBvciBqdXN0IHRleHQ/Pz8="

tls-auth:
        name: primary.example
        auth-domain-name: primary.example

zone:
        name: "catalog1.invalid"
        catalog: consumer
        catalog-member-pattern: "member-zone-config"

        request-xfr: 192.0.2.1@853 tsig-key.name primary.example
        allow-notify: 192.0.2.1 tsig-key.name

        allow-query: BLOCKED
```

The consumer zone `catalog1.invalid` is configured in the example as a secondary zone. It transfers the catalog from the primary at `192.0.2.1`. The transfer is mutually authenticated *using TSIG*.

The content of catalog zones are only relevant for the name servers handling those zones. They contain a list of zones that are served from the name servers and it is likely undesirable to expose that content. We have protected the zone against queries from third-parties by setting the `allow-query: BLOCKED` option. The transfer is protected against on-path eavesdroppers by doing it over authenticated TLS.

Note: Using privacy preserving option is RECOMMENDED for catalog zones (See RFC 9432 Sections 6 and 7).

Note: Catalog consumer zones do not need to be secondary, they may also process just zone files.

NSD supports the group property. Member zones from the catalog will be added with the pattern given by the group property of that member. If a member does not have a group property or its value is invalid or doesn't match a pattern, the pattern given by the `catalog-member-pattern:` option will be used.

4.2 Using nsd-control to get catalog zone status

The status of catalog zones and catalog member zones can be consulted with **`nsd-control zonestatus`**.

```
$ nsd-control zonestatus

zone:   catalog1.invalid
        catalog: consumer (serial: 1708341939, # members: 2)
        state: ok
        served-serial: "1708341939 since 2024-02-19T15:19:44"
        commit-serial: "1708341939 since 2024-02-19T15:19:44"
        wait: "3461 sec between attempts"

zone:   example.net
        pattern: member-zone-config
        catalog-member-id: a5b75379.zones.catalog1.invalid.
        state: ok
        served-serial: "2024013019 since 2024-02-19T14:25:43"
        commit-serial: "2024013019 since 2024-02-19T14:25:43"
        wait: "7195 sec between attempts"

zone:   example.org
        pattern: group1
        catalog-member-id: 96143f7d.zones.catalog1.invalid.
        state: ok
        served-serial: "2024013016 since 2024-02-19T14:18:10"
        commit-serial: "2024013016 since 2024-02-19T14:18:10"
        wait: "6544 sec between attempts"
```

The first `zone:` entry in the example output above shows the status our configured consumer zone `catalog1.invalid`. Besides its role (`consumer` or `producer`) it show the last SOA serial number that was successfully processed, and the number of member zones that were added by processing the consumer zone.

Note: If the catalog zone has become invalid and isn't processed anymore, **`nsd-control zonestatus`** will show the reason why.

`nsd-control zonestatus` will also show the `catalog-member-id` of catalog member zones. In the example output of **`nsd-control zonestatus`** above we can see that `example.net` and `example.org` are member zones from `catalog1.invalid`. Apparently the `example.net` member did not have a valid group property, because it has been added with the default `catalog-member-pattern: member-zone-config`.

4.3 Setting up NSD as a catalog producer

A catalog producer zone can be configured in NSD by setting the `catalog: producer` option. Unlike consumer zones, multiple producer zones may be configured. NSD creates the content of producer zones and therefore producer zones cannot be configured as secondary zones. Likewise, `zonefile:` options are only used to write the zone, never to read it.

An example catalog producer configuration could look like this:

```
server:
        interface: 192.0.2.1@853
        tls-port: 853
        tls-service-key: "primary.example.key.pem"
        tls-service-pem: "primary.example.cert.pem"

pattern:
        name: "group0"
        catalog-producer-zone: "catalog1.invalid"

pattern:
        name: "group1"
        catalog-producer-zone: "catalog1.invalid"

key:
        name: tsig-key.name
        algorithm: hmac-sha256
        secret: "SXMgdGhpcyBhIHNlY3JldCBvciBqdXN0IHRleHQ/Pz8="

zone:
        name: "catalog1.invalid"
        catalog: producer

        store-ixfr: yes
        provide-xfr: 203.0.113.1@853 tsig-key.name
        notify: 203.0.113.1 tsig-key.name

        allow-query: BLOCKED
```

The producer zone is configured as a primary and allows (in our example) transfer of the zone over TLS only. Also, just like with the consumer zone configuration example above, queries to this zone are `BLOCKED` to comply with RECOMMENDED privacy and security considerations. We also recommend - for primary zones in general - to serve *incremental* transfers (configured with `store-ixfr: yes`).

Zones can be added as member zones, by adding them to NSD with **`nsd-control addzone`** with a pattern that has the name of the producer zone as value of a `catalog-producer-zone:` option. In the example configuration above, patterns `"group0"` and `"group1"` both have that option.

Here is an example on how to do that:

```
$ nsd-control addzone example.net group0
ok
$ nsd-control addzone example.org group1
ok
```

Like with consumer zones and consumer member zones, **`nsd-control zonestatus`** can be used to check on the status of catalog producer zones and its members:

```
$ nsd-control zonestatus

zone:   catalog1.invalid
        catalog: producer (serial: 1708341939, # members: 2)
        state: primary

zone:   example.net
        pattern: group0
        catalog-member-id: a5b75379.zones.catalog1.invalid.
        state: primary

zone:   example.org
        pattern: group1
        catalog-member-id: 96143f7d.zones.catalog1.invalid.
        state: primary
```

Like with other zones added with **nsd-control addzone**, the member zones are persistently added to the zone list file (see the `zonelistfile:` configure option). The content of the catalog producer zone is not persistent and will be reconstructed from the member zone entries in the zone list file.

```
$ cat /var/db/nsd/zone.list
# NSD zone list
# name pattern
cat example.net group0 a5b75379
cat example.org group1 96143f7d
```

CHAPTER

FIVE

LOGGING

NSD does not provide any DNS logging. We believe that this is a separate task and has to be done independently from the core operation. This decision was taken in order to keep NSD focused and minimise its complexity. It is better to leave logging and tracing to separate dedicated tools. Do note, however, that NSD can be compiled with support for DNSTAP (see `nsd.conf(5)`).

The CAIDA dnsstat tool can easily be configured and/or modified to suit local statistics requirements without any danger of affecting the name server itself. We have run `dnsstat` on the same machine as NSD, and we would recommend using a multiprocessor if performance is an issue. Of course, `dnsstat` can also run on a separate machine that has MAC layer access to the network of the server.

The **`nsd-control`** tool can output some statistics, with **`nsd-control stats`** and **`nsd-control stats_noreset`**. In contrib/nsd_munin_ there is a Munin grapher plugin that uses it. The output of **`nsd-control stats`** is easy to read (text only) with scripts. The output values are documented on the **`nsd-control`** man page.

Another available tool is dnstop, which displays DNS statistics on your network.

CHAPTER

SIX

USING TRANSACTION SIGNATURE (TSIG)

NSD supports Transaction Signature (TSIG) for zone transfer and for notify sending and receiving, for any query to the server.

TSIG keys are based on shared secrets. These must be configured in the config file. To keep the secret in a separate file use `include:  "filename"` to include that file.

An example TSIG key named `sec1_key`:

```
key:
  name: "sec1_key"
  algorithm: hmac-md5
  secret: "6KM6qiKfwfEpamEq72HQdA=="
```

This key can then be used for any query to the NSD server. NSD will check if the signature is valid, and if so, return a signed answer. Unsigned queries will be given unsigned replies.

The key can be used to restrict the access control lists, for example to only allow zone transfer with the key, by listing the key name on the access control line.

```
# provides AXFR to the subnet when TSIG is used.
provide-xfr: 10.11.12.0/24 sec1_key
# allow only notifications that are signed
allow-notify: 192.168.0.0/16 sec1_key
```

If the TSIG key name is used in `notify` or `request-xfr` lines, the key is used to sign the request/notification messages.

CHAPTER

SEVEN

ZONE EXPIRY OF SECONDARY ZONES

NSD will keep track of the status of secondary zones, according to the timing values in the SOA record for the zone. When the refresh time of a zone is reached, the serial number is checked and a zone transfer is started if the zone has changed. Each primary server is tried in turn.

Primary zones cannot expire so they are always served. Zones are interpreted as primary zones if they have no `request-xfr:` statements in the config file.

After the expire timeout (from the SOA record at the zone apex) is reached, the zone becomes expired. NSD will return `SERVFAIL` for expired zones, and will attempt to perform a zone transfer from any of the primaries. After a zone transfer succeeds, or if the primary indicates that the SOA serial number is still the same, the zone returns to an operational state.

In contrast with e.g. BIND, the inception time for a secondary zone is stored on disk (in `xfrdfile: "xfrd.state"`), together with timeouts. If a secondary zone acquisition time is recent enough, NSD can start serving a zone immediately on loading, without querying the primary server.

If a secondary zone has expired and no primaries can be reached, but NSD should still serve the zone, delete the `xfrd.state` file, but leave the zone file for the zone intact. Make sure to stop NSD before you delete the file, as NSD writes it on exit. Upon loading NSD will treat the zone file that you as operator have provided as recent and will serve the zone. Even though NSD will start to serve the zone immediately, the zone will expire after the timeout is reached again. NSD will also attempt to confirm that you have provided the correct data by polling the primaries. So when the primary servers come back up, it will transfer the updated zone within <retry timeout from SOA> seconds.

It is possible to provide zone files for both primary and secondary zones via alternative means (say from email or rsync). Reload with SIGHUP or **`nsd-control reload`** to read the new zone file contents into the name database. When this is done the new zone will be served. For primary zones, NSD will issue notifications to all configured `notify:` targets. For secondary zones the above happens; NSD attempts to validate the zone from the primary (checking its SOA serial number).

CHAPTER EIGHT

INTERFACES

NSD will by default bind itself to the system default interface and service IPv4 and if available also IPv6. It is possible to service only IPv4 or IPv6 using the *-4*, *-6* command line options, or the `ip4-only` and `ip6-only` config file options.

The command line option *-a* and config file option ip-address can be given to bind to specific interfaces. Multiple interfaces can be specified, which is useful for two reasons:

- The specific interface bound will result in the OS bypassing routing tables for the interface selection. This results in a small performance gain. It is not the performance gain that is the problem: sometimes the routing tables can give the wrong answer, see the next point.
- The answer will be routed via the interface the query came from. This makes sure that the return address on the DNS replies is the same as the query was sent to. Many resolvers require the source address of the replies to be correct. The `ip-address:` option is easier than configuring the OS routing table to return the DNS replies via the correct interface.

The above means that even for systems with multiple interfaces where you intend to provide DNS service to all interfaces, it is prudent to specify all the interfaces as `ip-address` config file options.

With the config file option `ip-transparent` you can allow NSD to bind to non-local addresses.

CHAPTER NINE

TUNING

In version 4.3.0 of NSD, additional functionality was added to increase performance even more. Most notably, this includes processor affinity.

NSD is performant by design because it matters when operators serve hundreds of thousands or even millions of queries per second. We strive to make the right choices by default, like enabling the use of `libevent` at the configure stage to ensure the most efficient event mechanism is used on a given platform. e.g. `epoll` on Linux and `kqueue` on FreeBSD. Switches are available for operators who know the implementation on their system behaves correctly, like enabling the use of `recvmmsg` at the configure stage (`--enable-recvmmsg`) to read multiple messages from a socket in one system call.

By default NSD forks (only) one server. Modern computer systems however, may have more than one processor, and usually have more than one core per processor. The easiest way to scale up performance is to simply fork more servers by configuring server-count: to match the number of cores available in the system so that more queries can be answered simultaneously. If the operating system supports it, ensure `reuseport:` is set to `yes` to distribute incoming packets evenly across server processes to balance the load.

A couple of other options that the operator may want to consider:

1. Memory usage can be lowered (around 50%) by using zone files and disable the on-disk database by setting `database: ""`.
2. TCP capacity can be significantly increased by setting `tcp-count: 1000` and `tcp-timeout: 3`. Set `tcp-reject-overflow: yes` to prevent the kernel connection queue from growing.

9.1 Processor Affinity

The aforementioned settings provide an easy way to increase performance without the need for in-depth knowledge of the hardware. For operators that require even more throughput `cpu-affinity` is available.

The operating system's scheduling-algorithm determines which core a given task is allocated to. Processors build up state — e.g. by keeping frequently accessed data in cache memory — for the task that it is currently executing. Whenever a task switches cores, performance is degraded because the core it switched to has yet to build up said state. While this scheduling-algorithm works just fine for general-purpose computing, operators may want to designate a set of cores for best performance. The `cpu-affinity` family of configuration options was added to NSD specifically for that purpose.

Processor affinity is currently supported on Linux and FreeBSD. Other operating systems may be supported in the future, but not all operating systems that can run NSD support CPU pinning. To fully benefit from this feature, one must first determine which cores should be allocated to NSD. This requires some knowledge of the underlying hardware, but generally speaking every process should run on a dedicated core and the use of Hyper-Threading cores should be avoided to prevent resource contention. List every core designated to NSD in `cpu-affinity` and bind each server process to a specific core using `server-<N>-cpu-affinity` and `xfrd-cpu-affinity` to improve L1/L2 cache hit rates and reduce pipeline stalls/flushes.

```
server:
  server-count: 2
  cpu-affinity: 0 1 2
  server-1-cpu-affinity: 0
  server-2-cpu-affinity: 1
  xfrd-cpu-affinity: 2
```

9.2 Partition Sockets

`ip-address:` options in the `server:` clause can be configured per server or set of servers. Sockets configured for a specific server are closed by other servers on startup. This improves performance if a large number of sockets are scanned using `select/poll` and avoids waking up multiple servers when a packet comes in, known as the thundering herd problem. Though both problems are solved using a modern kernel and a modern I/O event mechanism, there is one other reason to partition sockets, explained below.

```
server:
  ip-address: 192.0.2.1 servers=1
```

9.3 Bind to Device

`ip-address:` options in the server: clause can now also be configured to bind directly to the network interface device on Linux (`bindtodevice=yes`) and to use a specific routing table on FreeBSD (`setfib=<N>`). These were added to ensure UDP responses go out over the same interface the query came in on if there are multiple interfaces configured on the same subnet, but there may be some performance benefits as well as the kernel does not have to go through the network interface selection process.

```
server:
  ip-address: 192.0.2.1 bindtodevice=yes setfib=<N>
```

Note: FreeBSD does not create extra routing tables on demand. Consult the FreeBSD Handbook, forums, etc. for information on how to configure multiple routing tables.

9.4 Combining Options

Field tests have shown best performance is achieved by combining the aforementioned options so that each network interface is essentially bound to a specific core. To do so, use one IP address per server process, pin that process to a designated core and bind directly to the network interface device.

```
server:
  server-count: 2
  cpu-affinity: 0 1 2
  server-1-cpu-affinity: 0
  server-2-cpu-affinity: 1
  xfrd-cpu-affinity: 2
  ip-address: 192.0.2.1 servers=1 bindtodevice=yes setfib=1
  ip-address: 192.0.2.2 servers=2 bindtodevice=yes setfib=2
```

The above snippet serves as an example on how to use the configuration options. Which cores, IP addresses and routing tables are best used depends entirely on the hardware and network layout. Be sure to test extensively before using the options.

CHAPTER TEN

NSD(8)

10.1 Synopsis

nsd [*-4*] [*-6*] [*-a* `ip-address[@port]`] [*-c* `configfile`] [*-d*] [*-f* `database`] [*-h*] [*-i* `identity`] [*-I* `nsid`] [*-l* `logfile`] [*-N* `server-count`] [*-n* `noncurrent-tcp-count`] [*-P* `pidfile`] [*-p* `port`] [*-s* `seconds`] [*-t* `chrootdir`] [*-u* `username`] [*-V* `level`] [*-v*]

10.2 Description

NSD is a complete implementation of an authoritative DNS nameserver. Upon startup, **NSD** will read the database specified with *-f* `database` argument and put itself into background and answers queries on port 53 or a different port specified with *-p* `port` option. The database is created if it does not exist. By default, **NSD** will bind to all local interfaces available. Use the *-a* `ip-address[@port]` option to specify a single particular interface address to be bound. If this option is given more than once, **NSD** will bind its UDP and TCP sockets to all the specified ip-addresses separately. If IPv6 is enabled when **NSD** is compiled an IPv6 address can also be specified.

10.3 Options

All the options can be specified in the configfile (*-c* argument), except for the *-v* and *-h* options. If options are specified on the commandline, the options on the commandline take precedence over the options in the configfile.

Normally **NSD** should be started with the *nsd-control(8)* `start` command invoked from a `/etc/rc.d/nsd.sh` script or similar at the operating system startup.

-4

Only listen to IPv4 connections.

-6

Only listen to IPv6 connections.

-a `ip-address[@port]`

Listen to the specified ip-address. The ip-address must be specified in numeric format (using the standard IPv4 or IPv6 notation). Optionally, a port number can be given. This flag can be specified multiple times to listen to multiple IP addresses. If this flag is not specified, **NSD** listens to the wildcard interface.

-c `configfile`

Read specified *configfile* instead of the default `/etc/nsd/nsd.conf`. For format description see *nsd.conf(5)*.

`-d`

Do not fork, stay in the foreground.

`-f database`

Use the specified *database* instead of the default of `/var/db/nsd/nsd.db`. If a `zonesdir:` is specified in the config file this path can be relative to that directory.

`-h`

Print help information and exit.

`-i identity`

Return the specified *identity* when asked for *CH TXT ID.SERVER* (This option is used to determine which server is answering the queries when they are anycast). The default is the name returned by gethostname(3).

`-I nsid`

Add the specified *nsid* to the EDNS section of the answer when queried with an NSID EDNS enabled packet. As a sequence of hex characters or with ascii_ prefix and then an ascii string.

`-l logfile`

Log messages to the specified logfile. The default is to log to stderr and syslog. If a `zonesdir:` is specified in the config file this path can be relative to that directory.

`-N count`

Start count `NSD` servers. The default is 1. Starting more than a single server is only useful on machines with multiple CPUs and/or network adapters.

`-n number`

The maximum number of concurrent TCP connection that can be handled by each server. The default is 100.

`-P pidfile`

Use the specified *pidfile* instead of the platform specific default, which is mostly `/var/run/nsd.pid`. If a `zonesdir:` is specified in the config file, this path can be relative to that directory.

`-p port`

Answer the queries on the specified *port*. Normally this is port 53.

`-s seconds`

Produce statistics dump every *seconds* seconds. This is equal to sending *SIGUSR1* to the daemon periodically.

`-t chroot`

Specifies a directory to *chroot* to upon startup. This option requires you to ensure that appropriate *syslogd(8)* socket (e.g. *chrootdir* /dev/log) is available, otherwise `NSD` won't produce any log output.

`-u username`

Drop user and group privileges to those of *username* after binding the socket. The *username* must be one of: username, id, or id.gid. For example: nsd, 80, or 80.80.

`-V level`

This value specifies the verbosity level for (non-debug) logging. Default is 0.

`-v`

Print the version number of `NSD` to standard error and exit.

`NSD` reacts to the following signals:

SIGTERM
Stop answering queries, shutdown, and exit normally.

SIGHUP Reload.
Scans zone files and if changed (mtime) reads them in. Also reopens the logfile (assists logrotation).

SIGUSR1
Dump BIND8-style statistics into the log. Ignored otherwise.

10.4 Files

/var/db/nsd/nsd.db
default `NSD` database

/var/run/nsd.pid
the process id of the name server.

/etc/nsd/nsd.conf
default `NSD` configuration file

10.5 Diagnostics

`NSD` will log all the problems via the standard *syslog(8)* daemon facility, unless the *`-d`* option is specified.

10.6 See Also

`nsd.conf(5)`, *`nsd-checkconf(8)`*, *`nsd-control(8)`*

CHAPTER

ELEVEN

NSD-CHECKCONF(8)

11.1 Synopsis

nsd-checkconf *-v -f -h* [*-o* `option`] [*-z* `zonename`] [*-p* `pattern`] [*-s* `keyname`] [*-t* `tlsauthname`] configfile

11.2 Description

nsd-checkconf reads a configuration file. It prints parse errors to standard error, and performs additional checks on the contents. The configfile format is described in *nsd.conf(5)*.

The utility of this program is to check a config file for errors before using it in *nsd(8)*. This program can also be used for shell scripts to access the nsd config file, using the *-o* and *-z* options.

11.3 Options

-v

After reading print the options to standard output in configfile format. Without this option, only success or parse errors are reported.

-f

Print full pathname when used with files, like with *-o* pidfile. This includes the chroot in the way it is applied to the pidfile.

-h

Print usage help information and exit.

-o `option`

Return only this option from the config file. This option can be used in conjunction with the *-z* and the *-p* option, or without them to query the server: section. The special value zones prints out a list of configured zones. The special value patterns prints out a list of configured patterns.

This option can be used to parse the config file from the shell. If the *-z* option is given, but the *-o* option is not given, nothing is printed.

-s `keyname`

Prints the key secret (base64 blob) configured for this key in the config file. Used to help shell scripts parse the config file.

-t `tls-auth`
: Prints the authentication domain name configured for this tls-auth clause in the config file. Used to help shell scripts parse the config file.

-p `pattern`
: Return the option specified with *-o* for the given pattern name.

-z `zonename`
: Return the option specified with *-o* for zone `zonename`.

If this option is not given, the server section of the config file is used.

The *-o*, *-s* and *-z* option print configfile options to standard output.

11.4 Files

/etc/nsd/nsd.conf
default NSD configuration file

11.5 See Also

nsd(8), *nsd.conf(5)*, *nsd.control(8)*

CHAPTER

TWELVE

NSD-CHECKZONE(8)

12.1 Synopsis

`nsd-checkzone` [*`-h`*] zonename zonefile

12.2 Description

`nsd-checkzone` reads a DNS zone file and checks it for errors. It prints errors to stderr. On failure it exits with nonzero exit status.

This is used to check files before feeding them to the *`nsd(8)`* daemon.

12.3 Options

`-h`

Print usage help information and exit.

zonename

The name of the zone to check, eg. "example.com".

zonefile

The file to read, eg. `zones/example.com.zone.signed`.

`-p`

Print the zone contents to stdout if the zone is ok. This prints the contents as it has been parsed, not literally a copy of the input, but as printed by the formatting routines in NSD, much like the nsd-control command write does.

12.4 See Also

`nsd(8)`, *`nsd-checkconf(8)`*

CHAPTER

THIRTEEN

NSD.CONF(5)

13.1 Synopsis

nsd.conf

13.2 Description

Nsd.conf is used to configure *nsd(8)*. The file format has attributes and values. Some attributes have attributes inside them. The notation is: `attribute:  value`.

Comments start with # and last to the end of line. Empty lines are ignored as is whitespace at the beginning of a line. Quotes can be used, for names with spaces, eg. "file name.zone".

Nsd.conf specifies options for the nsd server, zone files, primaries and secondaries.

13.3 Example

An example of a short nsd.conf file is below.

```
# Example.com nsd.conf file
# This is a comment.

server:
        server-count: 1 # use this number of cpu cores
        database: ""  # or use "/var/db/nsd/nsd.db"
        zonelistfile: "/var/db/nsd/zone.list"
        username: nsd
        logfile: "/var/log/nsd.log"
        pidfile: "/var/run/nsd.pid"
        xfrdfile: "/var/db/nsd/xfrd.state"

zone:
        name: example.com
        zonefile: /etc/nsd/example.com.zone

zone:
        # this server is master, 192.0.2.1 is the secondary.
        name: masterzone.com
```

(continues on next page)

(continued from previous page)

```
        zonefile: /etc/nsd/masterzone.com.zone
        notify: 192.0.2.1 NOKEY
        provide-xfr: 192.0.2.1 NOKEY

zone:
        # this server is secondary, 192.0.2.2 is master.
        name: secondzone.com
        zonefile: /etc/nsd/secondzone.com.zone
        allow-notify: 192.0.2.2 NOKEY
        request-xfr: 192.0.2.2 NOKEY
```

Then, use `kill -HUP` to reload changes from master zone files. And use `kill -TERM` to stop the server.

13.4 File Format

There must be whitespace between keywords. Attribute keywords end with a colon `':'`. An attribute is followed by its containing attributes, or a value.

At the top level only **server:**, **key:**, **pattern:**, **zone:**, **tls-auth:**, and **remote-control:** are allowed. These are followed by their attributes or a new top-level keyword. The **zone:** attribute is followed by zone options. The **server:** attribute is followed by global options for the NSD server. A **key:** attribute is used to define keys for authentication. The **pattern:** attribute is followed by the zone options for zones that use the pattern. A **tls-auth:** attribute is used to define credentials for authenticating an outgoing TLS connection used for XFR-over-TLS.

Files can be included using the **include:** directive. It can appear anywhere, and takes a single filename as an argument. Processing continues as if the text from the included file was copied into the config file at that point. If a chroot is used an absolute filename is needed (with the chroot prepended), so that the include can be parsed before and after application of the chroot (and the knowledge of what that chroot is). You can use `'*'` to include a wildcard match of files, e.g. `foo/nsd.d/*.conf`. Also `'?'`, `'{}'`, `'[]'`, and `'~'` work, see *glob(7)*. If no files match the pattern, this is not an error.

13.4.1 Server Options

The global options (if not overridden from the NSD commandline) are taken from the **server:** clause. There may only be one **server:** clause.

ip-address: <ip4 or ip6>[@port] [servers] [bindtodevice] [setfib]

NSD will bind to the listed ip-address. Can be given multiple times to bind multiple ip-addresses. Optionally, a port number can be given. If none are given NSD listens to the wildcard interface. Same as commandline option `-a`.

To limit which NSD server(s) listen on the given interface, specify one or more servers separated by whitespace after <ip>[@port]. Ranges can be used as a shorthand to specify multiple consecutive servers. By default every server will listen.

If an interface name is used instead of ip4 or ip6, the list of IP addresses associated with that interface is picked up and used at server start.

For servers with multiple IP addresses that can be used to send traffic to the internet, list them one by one, or the source address of replies could be wrong. This is because if the udp socket associates a source address of `0.0.0.0` then the kernel picks an ip-address with which to send to the internet, and it picks the wrong one. Typically needed for anycast instances. Use ip-transparent to be able to list addresses that turn on later (typical for certain load-balancing).

interface: <ip4 or ip6>[@port] [servers] [bindtodevice] [setfib]
Same as ip-address (for ease of compatibility with unbound.conf).

ip-transparent: <yes or no>
Allows NSD to bind to non local addresses. This is useful to have NSD listen to IP addresses that are not (yet) added to the network interface, so that it can answer immediately when the address is added. Default is no.

ip-freebind: <yes or no>
Set the IP_FREEBIND option to bind to nonlocal addresses and interfaces that are down. Similar to ip-transparent. Default is no.

reuseport: <yes or no>
Use the SO_REUSEPORT socket option, and create file descriptors for every server in the server-count. This improves performance of the network stack. Only really useful if you also configure a server-count higher than 1 (such as, equal to the number of cpus). The default is no. It works on Linux, but does not work on FreeBSD, and likely does not work on other systems.

send-buffer-size: <number>
Set the send buffer size for query-servicing sockets. Set to 0 to use the default settings.

receive-buffer-size: <number>
Set the receive buffer size for query-servicing sockets. Set to 0 to use the default settings.

debug-mode: <yes or no>
Turns on debugging mode for nsd, does not fork a daemon process. Default is no. Same as commandline option `-d`. If set to yes it does not fork and stays in the foreground, which can be helpful for commandline debugging, but is also used by certain server supervisor processes to ascertain that the server is running.

do-ip4: <yes or no>
If yes, NSD listens to IPv4 connections. Default yes.

do-ip6: <yes or no>
If yes, NSD listens to IPv6 connections. Default yes.

database: <filename>
By default '/var/db/nsd/nsd.db' is used. The specified file is used to store the compiled zone information. Same as commandline option `-f`. If set to `""` then no database is used. This uses less memory but zone updates are not (immediately) spooled to disk.

zonelistfile: <filename>
By default `/var/db/nsd/zone.list` is used. The specified file is used to store the dynamically added list of zones. The list is written to by NSD to add and delete zones. It is a text file with a zone-name and pattern-name on each line. This file is used for the nsd-control addzone and delzone commands.

identity: <string>
Returns the specified identity when asked for `CH TXT ID.SERVER`. Default is the name as returned by *gethostname(3)*. Same as commandline option `-i`. See hide-identity to set the server to not respond to such queries.

version: <string>
Returns the specified version string when asked for `CH TXT version.server`, and `version.bind` queries. Default is the compiled package version. See hide-version to set the server to not respond to such queries.

nsid: <string>
Add the specified nsid to the EDNS section of the answer when queried with an NSID EDNS enabled packet. As a sequence of hex characters or with ascii_ prefix and then an ascii string. Same as commandline option `-I`.

logfile: <filename>
Log messages to the logfile. The default is to log to stderr and syslog (with facility LOG_DAEMON). Same as commandline option `-l`.

log-only-syslog: <yes or no>
Log messages only to syslog. Useful with systemd so that print to stderr does not cause duplicate log strings in journald. Before syslog has been opened, the server uses stderr. Stderr is also used if syslog is not available. Default is no.

server-count: <number>
Start this many NSD servers. Default is 1. Same as commandline option `-N`.

cpu-affinity: <number> <number> ...
Overall CPU affinity for NSD server(s). Default is no affinity.

server-N-cpu-affinity: <number>
Bind NSD server specified by N to a specific core. Default is to have affinity set to every core specified in cpu-affinity. This setting only takes effect if cpu-affinity is enabled.

xfrd-cpu-affinity: <number>
Bind xfrd to a specific core. Default is to have affinity set to every core specified in cpu-affinity. This setting only takes effect if cpu-affinity is enabled.

tcp-count: <number>
The maximum number of concurrent, active TCP connections by each server. Default is 100. Same as commandline option `-n`.

tcp-reject-overflow: <yes or no>
If set to yes, TCP connections made beyond the maximum set by tcp-count will be dropped immediately (accepted and closed). Default is no.

tcp-query-count: <number>
The maximum number of queries served on a single TCP connection. Default is 0, meaning there is no maximum.

tcp-timeout: <number>
Overrides the default TCP timeout. This also affects zone transfers over TCP. The default is 120 seconds.

tcp-mss: <number>
Maximum segment size (MSS) of TCP socket on which the server responds to queries. Value lower than common MSS on Ethernet (1220 for example) will address path MTU problem. Note that not all platform supports socket option to set MSS (TCP_MAXSEG). Default is system default MSS determined by interface MTU and negotiation between server and client.

outgoing-tcp-mss: <number>
Maximum segment size (MSS) of TCP socket for outgoing XFR request to other namesevers. Value lower than common MSS on Ethernet (1220 for example) will address path MTU problem. Note that not all platform supports socket option to set MSS (TCP_MAXSEG). Default is system default MSS determined by interface MTU and negotiation between NSD and other servers.

ipv4-edns-size: <number>
Preferred EDNS buffer size for IPv4. Default 1232.

ipv6-edns-size: <number>
Preferred EDNS buffer size for IPv6. Default 1232.

pidfile: <filename>
Use the pid file instead of the platform specific default, usually `/var/run/nsd.pid`. Same as commandline option `-P`. With `""` there is no pidfile, for some startup management setups, where a pidfile is not useful to have.

port: <number>
Answer queries on the specified port. Default is 53. Same as commandline option `-p`.

statistics: <number>
If not present no statistics are dumped. Statistics are produced every number seconds. Same as commandline option `-s`.

chroot: <directory>
NSD will chroot on startup to the specified directory. Note that if elsewhere in the configuration you specify an absolute pathname to a file inside the chroot, you have to prepend the chroot path. That way, you can switch the chroot option on and off without having to modify anything else in the configuration. Set the value to `""` (the empty string) to disable the chroot. By default `""` is used. Same as commandline option *-t*.

username: <username>
After binding the socket, drop user privileges and assume the username. Can be username, id or id.gid. Same as commandline option *-u*.

zonesdir: <directory>
Change the working directory to the specified directory before accessing zone files. Also, NSD will access **database**, **zonelist-file**, **logfile**, **pidfile**, **xfrdfile**, **xfrdir**, **server-key-file**, **server-cert-file**, **control-key-file** and **control-cert-file** relative to this directory. Set the value to `""` (the empty string) to disable the change of working directory. By default `"/etc/nsd"` is used.

difffile: <filename>
Ignored, for compatibility with NSD3 config files.

xfrdfile: <filename>
The soa timeout and zone transfer daemon in NSD will save its state to this file. State is read back after a restart. The state file can be deleted without too much harm, but timestamps of zones will be gone. If it is configured as `""`, the state file is not used, all slave zones are checked for updates upon startup. For more details see the section on zone expiry behavior of NSD. Default is `/var/db/nsd/xfrd.state`.

xfrdir: <directory>
The zone transfers are stored here before they are processed. A directory is created here that is removed when NSD exits. Default is `/tmp`.

xfrd-reload-timeout: <number>
If this value is -1, xfrd will not trigger a reload after a zone transfer. If positive xfrd will trigger a reload after a zone transfer, then it will wait for the number of seconds before it will trigger a new reload. Setting this value throttles the reloads to once per the number of seconds. The default is 1 second.

verbosity: <level>
This value specifies the verbosity level for (non-debug) logging. Default is 0. 1 gives more information about incoming notifies and zone transfers. 2 lists soft warnings that are encountered. 3 prints more information.

Verbosity 0 will print warnings and errors, and other events that are important to keep NSD running.

Verbosity 1 prints additionally messages of interest. Successful notifies, successful incoming zone transfer (the zone is updated), failed incoming zone transfers or the inability to process zone updates.

Verbosity 2 prints additionally soft errors, like connection resets over TCP. And notify refusal, and axfr request refusals.

hide-version: <yes or no>
Prevent NSD from replying with the version string on CHAOS class queries. Default is no.

hide-identity: <yes or no>
Prevent NSD from replying with the identity string on CHAOS class queries. Default is no.

drop-updates: <yes or no>
If set to yes, drop received packets with the UPDATE opcode. Default is no.

use-systemd: <yes or no>
This option is deprecated and ignored. If compiled with libsystemd, NSD signals readiness to systemd and use of the option is not necessary.

log-time-ascii: <yes or no>
Log time in ascii, if "no" then in seconds epoch. Default is yes. This chooses the format when logging to file.

The print- out via syslog has a timestamp formatted by syslog.

round-robin: <yes or no>
Enable round robin rotation of records in the answer. This changes the order of records in the answer and this may balance load across them. The default is no.

minimal-responses: <yes or no>
Enable minimal responses for smaller answers. This makes pack- ets smaller. Extra data is only added for referrals, when it is really necessary. This is different from the –enable-minimal-responses configure time option, that reduces packets, but ex- actly to the fragmentation length, the nsd.conf option reduces packets as small as possible. The default is no.

confine-to-zone: <yes or no>
If set to yes, additional information will not be added to the response if the apex zone of the additional information does not match the apex zone of the initial query (E.G. CNAME resolution). Default is no.

refuse-any: <yes or no>
Refuse queries of type ANY. This is useful to stop query floods trying to get large responses. Note that rrl ratelimiting also has type ANY as a ratelimiting type. It sends truncation in response to UDP type ANY queries, and it allows TCP type ANY queries like normal. The default is no.

zonefiles-check: <yes or no>
Make NSD check the mtime of zone files on start and sighup. If you disable it it starts faster (less disk activity in case of a lot of zones). The default is yes. The nsd-control reload command reloads zone files regardless of this option.

zonefiles-write: <seconds>
Write changed secondary zones to their zonefile every N seconds. If the zone (pattern) configuration has "" zonefile, it is not written. Zones that have received zone transfer updates are written to their zonefile. Default is 0 (disabled) when there is a database, and 3600 (1 hour) when database is "". The database also commits zone transfer contents. You can configure it away from the default by putting the config statement for zone-files-write: after the database: statement in the config file.

rrl-size: <numbuckets>
This option gives the size of the hashtable. Default 1000000. More buckets use more memory, and reduce the chance of hash collisions.

rrl-ratelimit: <qps>
The max qps allowed (from one query source). Default is on (with a suggested 200 qps). If set to 0 then it is disabled (unlimited rate), also set the whitelist-ratelimit to 0 to disable rate-limit processing. If you set verbosity to 2 the blocked and unblocked subnets are logged. Blocked queries are blocked and some receive TCP fallback replies. Once the rate limit is reached, NSD begins dropping responses. However, one in every “rrl-slip” number of responses is allowed, with the TC bit set. If slip is set to 2, the outgoing response rate will be halved. If it’s set to 3, the outgoing response rate will be one-third, and so on. If you set rrl-slip to 10, traffic is reduced to 1/10th. Ratelimit options rrl-ratelimit, rrl-size and rrl-whitelist-ratelimit are updated when nsd-control reconfig is done (also the zone-specific ratelimit options are updated).

rrl-slip: <numpackets>
This option controls the number of packets discarded before we send back a SLIP response (a response with “truncated” bit set to one). 0 disables the sending of SLIP packets, 1 means every query will get a SLIP response. Default is 2, cuts traffic in half and legit users have a fair chance to get a +TC response.

rrl-ipv4-prefix-length: <subnet>
IPv4 prefix length. Addresses are grouped by netblock. Default 24.

rrl-ipv6-prefix-length: <subnet>
IPv6 prefix length. Addresses are grouped by netblock. Default 64.

rrl-whitelist-ratelimit: <qps>
The max qps for query sorts for a source, which have been whitelisted. Default on (with a suggested 2000 qps).

With the rrl-whitelist option you can set specific queries to receive this qps limit instead of the normal limit. With the value 0 the rate is unlimited.

answer-cookie: <yes or no>
Enable to answer to requests containig DNS Cookies as specified in **RFC 7873**. Default is no.

cookie-secret: <128 bit hex string>
Servers in an anycast deployment need to be able to verify each other's DNS Server Cookies. For this they need to share the secret used to construct and verify the DNS Cookies. Default is a 128 bits random secret generated at startup time. This option is ignored if a **cookie-secret-file** is present. In that case the secrets from that file are used in DNS Cookie calculations.

cookie-secret-file: <filename>
File from which the secrets are read used in DNS Cookie calculations. When this file exists, the secrets in this file are used and the secret specified by the **cookie-secret** option is ignored. Default is `/etc/nsd/nsd_cookiesecrets.txt`

The content of this file must be manipulated with the **add_cookie_secret**, **drop_cookie_secret** and **activate_cookie_secret** commands to the *nsd-control(8)* tool. Please see that manpage how to perform a safe cookie secret rollover.

tls-service-key: <filename>
If enabled, the server provides TLS service on TCP sockets with the TLS service port number. The port number (853) is configured with tls-port. To turn it on, create an interface: option line in config with @port appended to the IP-address. This creates the extra socket on which the DNS over TLS service is provided.

The file is the private key for the TLS session. The public certificate is in the tls-service-pem file. Default is `""`, turned off. Requires a restart (a reload is not enough) if changed, because the private key is read while root permissions are held and before chroot (if any).

tls-service-pem: <filename>
The public key certificate pem file for the tls service. Default is `""`, turned off.

tls-service-ocsp: <filename>
The ocsp pem file for the tls service, for OCSP stapling. Default is `""`, turned off. An external process prepares and updates the OCSP stapling data. Like this,

```
openssl ocsp -no_nonce \
-respout /path/to/ocsp.pem \
-CAfile /path/to/ca_and_any_intermediate.pem \
-issuer /path/to/direct_issuer.pem \
-cert /path/to/cert.pem \
-url "$( openssl x509 -noout -text -in /path/to/cert.pem |
grep 'OCSP - URI:' | cut -d: -f2,3 )"
```

tls-port: <number>
The port number on which to provide TCP TLS service, default is 853, only interfaces configured with that port number as @number get DNS over TLS service.

tls-cert-bundle: <filename>
If null or `""`, the default verify locations are used. Set it to the certificate bundle file, for example `"/etc/pki/tls/certs/ca-bundle.crt"`. These certificates are used for authenticating Transfer over TLS (XoT) connections.

13.4.2 Remote Control

The **remote-control:** clause is used to set options for using the `nsd-control(8)` tool to give commands to the running NSD server. It is disabled by default, and listens for localhost by default. It uses TLS over TCP where the server and client authenticate to each other with self-signed certificates. The self-signed certificates can be generated with the *nsd-control-setup* tool. The key files are read by NSD before the chroot and before dropping user permissions, so they can be outside the chroot and readable by the superuser only.

control-enable: <yes or no>
Enable remote control, default is no.

control-interface: <ip4 or ip6 | interface name | absolute path>
NSD will bind to the listed addresses to service control requests (on TCP). Can be given multiple times to bind multiple ip-addresses. Use 0.0.0.0 and ::0 to service the wildcard interface. If none are given NSD listens to the localhost 127.0.0.1 and ::1 interfaces for control, if control is enabled with control-enable.

If an interface name is used instead of ip4 or ip6, the list of IP addresses associated with that interface is picked up and used at server start.

With an absolute path, a unix local named pipe is used for control. The file is created with user and group that is configured and access bits are set to allow members of the group access. Further access can be controlled by setting permissions on the directory containing the control socket file. The key and cert files are not used when control is via the named pipe, because access control is via file and directory permission.

control-port: <number>
The port number for remote control service. 8952 by default.

server-key-file: <filename>
Path to the server private key, by default `/etc/nsd/nsd_server.key`. This file is generated by the nsd-control-setup utility. This file is used by the nsd server, but not by *nsd-control*.

server-cert-file: <filename>
Path to the server self signed certificate, by default `/etc/nsd/nsd_server.pem`. This file is generated by the *nsd-control-setup* utility. This file is used by the nsd server, and also by *nsd-control*.

control-key-file: <filename>
Path to the control client private key, by default `/etc/nsd/nsd_control.key`. This file is generated by the *nsd-control-setup* utility. This file is used by *nsd-control*.

control-cert-file: <filename>
Path to the control client certificate, by default `/etc/nsd/nsd_control.pem`. This certificate has to be signed with the server certificate. This file is generated by the *nsd-control-setup* utility. This file is used by *nsd-control*.

13.4.3 Pattern Options

The **pattern:** clause is used to denote a set of options to apply to some zones. The same zone options as for a zone are allowed.

name: <string>
The name of the pattern. This is a (case sensitive) string. The pattern names that start with "_implicit_" are used internally for zones that have no pattern (they are defined in *nsd.conf* directly).

include-pattern: <pattern-name>
The options from the given pattern are included at this point in this pattern. The referenced pattern must be defined above this one.

<zone option>: <value>
The zone options such as **zonefile**, **allow-query**, **allow-notify**, **request-xfr**, **allow-axfr-fallback**, **notify**, **notify-**

retry, **provide-xfr**, **zonestats**, and **outgoing-interface** can be given. They are applied to the patterns and zones that include this pattern.

13.4.4 Zone Options

For every zone the options need to be specified in one **zone:** clause. The access control list elements can be given multiple times to add multiple servers. These elements need to be added explicitly.

For zones that are configured in the *nsd.conf* config file their settings are hardcoded (in an implicit pattern for themselves only) and they cannot be deleted via delzone, but remove them from the config file and repattern.

name: <string>
The name of the zone. This is the domain name of the apex of the zone. May end with a `'.'` (in FQDN notation). For example "example.com", "sub.example.net.". This attribute must be present in each zone.

zonefile: <filename>
The file containing the zone information. If this attribute is present it is used to read and write the zone contents. If the attribute is absent it prevents writing out of the zone.

The string is processed so that one string can be used (in a pattern) for a lot of different zones. If the label or character does not exist the percent-character is replaced with a period for output (i.e. for the third character in a two letter domain name).

%s is replaced with the zone name.

%1 is replaced with the first character of the zone name.

%2 is replaced with the second character of the zone name.

%3 is replaced with the third character of the zone name.

%z is replaced with the toplevel domain name of the zone.

%y is replaced with the next label under the toplevel domain.

%x is replaced with the next-next label under the toplevel domain.

allow-query: <ip-spec> <key-name | NOKEY | BLOCKED>
Access control list. When at least one **allow-query** option is specified, then the in the **allow-query** options specified addresses are are allowed to query the server for the zone. Queries from unlisted or specifically BLOCKED addresses are discarded. If NOKEY is given no TSIG signature is required. BLOCKED supersedes other entries, other entries are scanned for a match in the order of the statements. Without **allow-query** options, queries are allowed from any IP address without TSIG key (which is the default).

The ip-spec is either a plain IP address (IPv4 or IPv6), or can be a subnet of the form `1.2.3.4/24`, or masked like `1.2.3.4&255.255.255.0` or a range of the form `1.2.3.4-1.2.3.25`. Note the ip-spec ranges do not use spaces around the `/`, `&`, `@` and `-` symbols.

allow-notify: <ip-spec> <key-name | NOKEY | BLOCKED>
Access control list. The listed (primary) address is allowed to send notifies to this (secondary) server. Notifies from unlisted or specifically BLOCKED addresses are discarded. If NOKEY is given no TSIG signature is required. BLOCKED supersedes other entries, other entries are scanned for a match in the order of the statements.

The ip-spec is either a plain IP address (IPv4 or IPv6), or can be a subnet of the form `1.2.3.4/24`, or masked like `1.2.3.4&255.255.255.0` or a range of the form `1.2.3.4-1.2.3.25`. A port number can be added using a suffix of @number, for example `1.2.3.4@5300` or `1.2.3.4/24@5300` for port 5300. Note the ip-spec ranges do not use spaces around the `/`, `&`, `@` and `-` symbols.

request-xfr: [AXFR|UDP] <ip-address> <key-name | NOKEY> [tls-auth-name]
Access control list. The listed address (the master) is queried for AXFR/IXFR on update. A port number can be

added using a suffix of @number, for example `1.2.3.4@5300`. The specified key is used during AXFR/IXFR. If tls-auth-name is included, the specified tls-auth clause will be used to perform authenticated XFR-over-TLS.

If the AXFR option is given, the server will not be contacted with IXFR queries but only AXFR requests will be made to the server. This allows an NSD secondary to have a master server that runs NSD. If the AXFR option is left out then both IXFR and AXFR requests are made to the master server.

If the UDP option is given, the secondary will use UDP to transmit the IXFR requests. You should deploy TSIG when allowing UDP transport, to authenticate notifies and zone transfers. Otherwise, NSD is more vulnerable for Kaminsky-style attacks. If the UDP option is left out then IXFR will be transmitted using TCP.

If a tls-auth-name is given then TLS (by default on port 853) will be used for all zone transfers for the zone. If authentication of the master based on the specified tls-auth authentication information fails, the XFR request will not be sent. Support for TLS 1.3 is required for XFR-over-TLS.

allow-axfr-fallback: <yes or no>
This option should be accompanied by request-xfr. It (dis)allows NSD (as secondary) to fallback to AXFR if the primary name server does not support IXFR. Default is yes.

size-limit-xfr: <number>
This option should be accompanied by request-xfr. It specifies XFR temporary file size limit. It can be used to stop very large zone retrieval, that could otherwise use up a lot of memory and disk space. If this option is 0, unlimited. Default value is 0.

notify: <ip-address> <key-name | NOKEY>
Access control list. The listed address (a secondary) is notified of updates to this zone. A port number can be added using a suffix of @number, for example `1.2.3.4@5300`. The specified key is used to sign the notify. Only on secondary configurations will NSD be able to detect zone updates (as it gets notified itself, or refreshes after a time).

notify-retry: <number>
This option should be accompanied by notify. It sets the number of retries when sending notifies.

provide-xfr: <ip-spec> <key-name | NOKEY | BLOCKED>
Access control list. The listed address (a secondary) is allowed to request AXFR from this server. Zone data will be provided to the address. The specified key is used during AXFR. For unlisted or BLOCKED addresses no data is provided, requests are discarded. BLOCKED supersedes other entries, other entries are scanned for a match in the order of the statements. NSD provides AXFR for its secondaries, but IXFR is not implemented (IXFR is implemented for request-xfr, but not for provide-xfr).

The ip-spec is either a plain IP address (IPv4 or IPv6), or can be a subnet of the form `1.2.3.4/24`, or masked like `1.2.3.4&255.255.255.0` or a range of the form `1.2.3.4-1.2.3.25`. A port number can be added using a suffix of @number, for example `1.2.3.4@5300` or `1.2.3.4/24@5300` for port 5300. Note the ip-spec ranges do not use spaces around the the `/`, `&`, `@` and `-` symbols.

outgoing-interface: <ip-address>
Access control list. The listed address is used to request AXFR|IXFR (in case of a secondary) or used to send notifies (in case of a primary).

The ip-address is a plain IP address (IPv4 or IPv6). A port number can be added using a suffix of @number, for example `1.2.3.4@5300`.

max-refresh-time: <seconds>
Limit refresh time for secondary zones. This is the timer which checks to see if the zone has to be refetched when it expires. Normally the value from the SOA record is used, but this option restricts that value.

min-refresh-time: <seconds>
Limit refresh time for secondary zones.

max-retry-time: <seconds>
Limit retry time for secondary zones. This is the timer which retries after a failed fetch attempt for the zone.

Normally the value from the SOA record is used, followed by an exponential backoff, but this option restricts that value.

min-retry-time: <seconds>
Limit retry time for secondary zones.

min-expire-time: <seconds or refresh+retry+1>
Limit expire time for secondary zones. The value can be expressed either by a number of seconds, or the string "refresh+retry+1". With the latter the expire time will be lower bound to the refresh plus the retry value from the SOA record, plus 1. The refresh and retry values will be subject to the bounds configured with max-refresh-time, min-refresh-time, max-retry-time and min-retry-time if given.

zonestats: <name>
When compiled with `--enable-zone-stats` NSD can collect statistics per zone. This name gives the group where statistics are added to. The groups are output from nsd-control stats and stats_noreset. Default is `""`. You can use `"%s"` to use the name of the zone to track its statistics. If not compiled in, the option can be given but is ignored.

include-pattern: <pattern-name>
The options from the given pattern are included at this point. The referenced pattern must be defined above this zone.

rrl-whitelist: <rrltype>
This option causes queries of this rrltype to be whitelisted, for this zone. They receive the whitelist-ratelimit. You can give multiple lines, each enables a new rrltype to be whitelisted for the zone. Default has none whitelisted. The rrl-type is the query classification that the NSD RRL employs to make different types not interfere with one another. The types are logged in the loglines when a subnet is blocked (in verbosity 2). The RRL classification types are: nxdomain, error, referral, any, rrsig, wildcard, nodata, dnskey, positive, all.

multi-master-check: <yes or no>
Default no. If enabled, checks all masters for the last version. It uses the higher version of all the configured masters. Useful if you have multiple masters that have different version numbers served.

13.4.5 Key Declarations

The **key:** clause establishes a key for use in access control lists. It has the following attributes.

name: <string>
The key name. Used to refer to this key in the access control list. The key name has to be correct for tsig to work. This is because the key name is output on the wire.

algorithm: <string>
Authentication algorithm for this key. Such as hmac-md5, hmac-sha1, hmac-sha224, hmac-sha256, hmac-sha384 and hmac-sha512. Can also be abbreviated as 'sha1', 'sha256'. Default is sha256. Algorithms are only available when they were compiled in (available in the crypto library).

secret: <base64 blob>
The base64 encoded shared secret. It is possible to put the **secret:** declaration (and base64 blob) into a different file, and then to **include:** that file. In this way the key secret and the rest of the configuration file, which may have different security policies, can be split apart. The content of the secret is the agreed base64 secret content. To make it up, enter a password (its length must be a multiple of 4 characters, A-Za-z0-9), or use dev-random output through a base64 encode filter.

13.4.6 TLS Auth Declarations

The **tls-auth:** clause establishes authentication attributes to use when authenticating the far end of an outgoing TLS connection used in access control lists for XFR-over-TLS. It has the following attributes.

name: <string>
The tls-auth name. Used to refer to this TLS authentication information in the access control list.

auth-domain-name: <string>
The authentication domain name as defined in **RFC 8310**.

client-cert: <file name of clientcert.pem>
If you want to use mutual TLS authentication, this is where the client certificates can be configured that NSD uses to connect to the upstream server to download the zone. The client public key pem cert file can be configured here. Also configure a private key with client-key.

client-key: <file name of clientkey.key>
If you want to use mutual TLS authentication, the private key file can be configured here for the client authentication.

client-key-pw: <string>
If the client-key file uses a password to decrypt the key before it can be used, then the password can be specified here as a string. It is possible to include other config files with the include: option, and this can be used to move that sensitive data to another file, if you wish.

13.4.7 DNSTAP Logging Options

DNSTAP support, when compiled in, is enabled in the **dnstap:** section. This starts a collector process that writes the log information to the destination.

dnstap-enable: <yes or no>
If dnstap is enabled. Default no. If yes, it connects to the dnstap server and if any of the dnstap-log-..-messages options is enabled it sends logs for those messages to the server.

dnstap-socket-path: <file name>
Sets the unix socket file name for connecting to the server that is listening on that socket. Default is "`/var/run/nsd-dnstap.sock`".

dnstap-send-identity: <yes or no>
If enabled, the server identity is included in the log messages. Default is no.

dnstap-send-version: <yes or no>
If enabled, the server version if included in the log messages. Default is no.

dnstap-identity: <string>
The identity to send with messages, if `""` the hostname is used. Default is `""`.

dnstap-version: <string>
The version to send with messages, if `""` the package version is used. Default is `""`.

dnstap-log-auth-query-messages: <yes or no>
Enable to log auth query messages. Default is no. These are client queries to NSD.

dnstap-log-auth-response-messages: <yes or no>
Enable to log auth response messages. Default is no. These are responses from NSD to clients.

13.5 NSD Configuration for BIND9 Hackers

BIND9 is a name server implementation with its own configuration file format, *named.conf(5)*. BIND9 types zones as ‘Master’ or ‘Slave’.

13.5.1 Slave zones

For a slave zone, the master servers are listed. The master servers are queried for zone data, and are listened to for update notifications. In NSD these two properties need to be configured separately, by listing the master address in allow-notify and request-xfr statements.

In BIND9 you only need to provide allow-notify elements for any extra sources of notifications (i.e. the operators), NSD needs to have allow-notify for both masters and operators. BIND9 allows additional transfer sources, in NSD you list those as request-xfr.

Here is an example of a slave zone in BIND9 syntax.

```
# Config file for example.org options {
    dnssec-enable yes;
};

key tsig.example.org. {
        algorithm hmac-md5;
        secret "aaaaaabbbbbbccccccdddddd";
};

server 162.0.4.49 {
        keys { tsig.example.org. ; };
};

zone "example.org" {
        type slave;
        file "secondary/example.org.signed";
        masters { 162.0.4.49; };
};
```

For NSD, DNSSEC is enabled automatically for zones that are signed. The **dnssec-enable** statement in the options clause is not needed. In NSD keys are associated with an IP address in the access control list statement, therefore the **server{}** statement is not needed. Below is the same example in an NSD config file.

```
# Config file for example.org
key:
        name: tsig.example.org.
        algorithm: hmac-md5
        secret: "aaaaaabbbbbbccccccdddddd"

zone:
        name: "example.org"
        zonefile: "secondary/example.org.signed"
        # the master is allowed to notify and will provide zone data.
        allow-notify: 162.0.4.49 NOKEY
        request-xfr: 162.0.4.49 tsig.example.org.
```

Notice that the master is listed twice, once to allow it to send notifies to this slave server and once to tell the slave server where to look for updates zone data. More allow-notify and request-xfr lines can be added to specify more masters.

It is possible to specify extra allow-notify lines for addresses that are also allowed to send notifications to this slave server.

13.5.2 Master zones

For a master zone in BIND9, the slave servers are listed. These slave servers are sent notifications of updated and are allowed to request transfer of the zone data. In NSD these two properties need to be configured separately.

Here is an example of a master zone in BIND9 syntax.

```
zone "example.nl" {
        type master;
        file "example.nl";
};
```

In NSD syntax this becomes:

zone:

name: "example.nl" zonefile: "example.nl" # allow anybody to request xfr. provide-xfr: 0.0.0.0/0 NOKEY provide-xfr: ::0/0 NOKEY

# to list a slave server you would in general give # provide-xfr: 1.2.3.4 tsig-key.name. # notify: 1.2.3.4 NOKEY

13.5.3 Other

NSD is an authoritative only DNS server. This means that it is meant as a primary or secondary server for zones, providing DNS data to DNS resolvers and caches. BIND9 can function as an authoritative DNS server, the configuration options for that are compared with those for NSD in this section. However, BIND9 can also function as a resolver or cache. The configuration options that BIND9 has for the resolver or caching thus have no equivalents for NSD.

13.6 Files

/var/db/nsd/nsd.db
default **NSD** database

/etc/nsd/nsd.conf
default **NSD** configuration file

13.7 See Also

nsd(8), *nsd-checkconf(8)*, *nsd-control(8)*

13.8 Bugs

nsd.conf is parsed by a primitive parser, error messages may not be to the point.

CHAPTER

FOURTEEN

NSD-CONTROL(8)

14.1 Synopsis

nsd-control [*-c* cfgfile] [*-s* server] command

14.2 Description

nsd-control performs remote administration on the *nsd(8)* DNS server. It reads the configuration file, contacts the nsd server over TLS, sends the command and displays the result.

The available options are:

-h

Show the version and commandline option help.

-c cfgfile

The config file to read with settings. If not given the default config file `/etc/nsd/nsd.conf` is used.

-s server[@port]

IPv4 or IPv6 address of the server to contact. If not given, the address is read from the config file.

14.3 Commands

There are several commands that the server understands.

start

Start the server. Simply execs *nsd(8)*. The nsd executable is searched for in the **PATH** set in the environment. It is started with the config file specified using *-c* or the default config file.

stop

Stop the server. The server daemon exits.

reload [<zone>]

Reload zonefiles and reopen logfile. Without argument reads changed zonefiles. With argument reads the zonefile for the given zone and loads it.

reconfig

Reload nsd.conf and apply changes to TSIG keys and configuration patterns, and apply the changes to add and remove zones that are mentioned in the config. Other changes are not applied, such as listening ip address and port and chroot, also per-zone statistics are not applied. The pattern updates means that the configuration options for zones (request-xfr, zonefile, notify, . . .) are updated. Also new patterns are available for use with the addzone command.

repattern
Same as the reconfig option.

log_reopen
Reopen the logfile, for log rotate that wants to move the logfile away and create a new logfile. The log can also be reopened with kill -HUP (which also reloads all zonefiles).

status
Display server status. Exit code 3 if not running (the connection to the port is refused), 1 on error, 0 if running.

stats
Output a sequence of name=value lines with statistics information, requires NSD to be compiled with this option enabled.

stats_noreset
Same as stats, but does not zero the counters.

addzone <zone name> <pattern name>
Add a new zone to the running server. The zone is added to the zonelist file on disk, so it stays after a restart. The pattern name determines the options for the new zone. For slave zones a zone transfer is immediately attempted. For zones with a zonefile, the zone file is attempted to be read in.

delzone <zone name>
Remove the zone from the running server. The zone is removed from the zonelist file on disk, from the nsd.db file and from the memory. If it had a zonefile, this remains (but may be outdated). Zones configured inside nsd.conf itself cannot be removed this way because the daemon does not write to the nsd.conf file, you need to add such zones to the zonelist file to be able to delete them with the delzone command.

changezone <zone name> <pattern name>
Change a zone to use the pattern for options. The zone is deleted and added in one operation, changing it to use the new pattern for the zone options. Zones configured in nsd.conf cannot be changed like this, instead edit the nsd.conf (or the included file in nsd.conf) and reconfig.

addzones
Add zones read from stdin of nsd-control. Input is read per line, with name space patternname on a line. For bulk additions.

delzones
Remove zones read from stdin of nsd-control. Input is one name per line. For bulk removals.

write [<zone>]
Write zonefiles to disk, or the given zonefile to disk. Zones that have changed (via AXFR or IXFR) are written, or if the zonefile has not been created yet then it is created. Directory components of the zonefile path are created if necessary. With argument that zone is written if it was modified, without argument, all modified zones are written.

notify [<zone>]
Send NOTIFY messages to slave servers. Sends to the IP addresses configured in the 'notify:' lists for the master zones hosted on this server. Usually NSD sends NOTIFY messages right away when a master zone serial is updated. If a zone is given, notifies are sent for that zone. These slave servers are supposed to initiate a zone transfer request later (to this server or another master), this can be allowed via the 'provide-xfr:' acl list configuration. With argument that zone is processed, without argument, all zones are processed.

transfer [<zone>]
Attempt to update slave zones that are hosted on this server by contacting the masters. The masters are configured via 'request-xfr:' lists. If a zone is given, that zone is updated. Usually NSD receives a NOTIFY from the masters (configured via 'allow-notify:' acl list) that a new zone serial has to be transferred. For zones with no content,

NSD may have backed off from asking often because the masters did not respond, but this command will reset the backoff to its initial timeout, for frequent retries. With argument that zone is transferred, without argument, all zones are transferred.

force_transfer [<zone>]
Force update slave zones that are hosted on this server. Even if the master hosts the same serial number of the zone, a full AXFR is performed to fetch it. If you want to use IXFR and check that the serial number increases, use the 'transfer' command. With argument that zone is transferred, without argument, all zones are transferred.

zonestatus [<zone>]
Print state of the zone, the serial numbers and since when they have been acquired. Also prints the notify action (to which server), and zone transfer (and from which master) if there is activity right now. The state of the zone is printed as: 'master' (master zones), 'ok' (slave zone is up-to-date), 'expired' (slave zone has expired), 'refreshing' (slave zone has transfers active). The serial numbers printed are the 'served-serial' (currently active), the 'commit-serial' (is in reload), the 'notified-serial' (got notify, busy fetching the data). The serial numbers are only printed if such a serial number is available. With argument that zone is printed, without argument, all zones are printed.

serverpid
Prints the PID of the server process. This is used for statistics (and only works when NSD is compiled with statistics en- abled). This pid is not for sending unix signals, use the pid from nsd.pid for that, that pid is also stable.

verbosity <number>
Change logging verbosity.

print_tsig [<key_name>]
print the secret and algorithm for the TSIG key with that name. Or list all the tsig keys with their name, secret and algorithm.

update_tsig <name> <secret>
Change existing TSIG key with name to the new secret. The secret is a base64 encoded string. The changes are only in-memory and are gone next restart, for lasting changes edit the nsd.conf file or a file included from it.

add_tsig <name> <secret> [algo]
Add a new TSIG key with the given name, secret and algorithm. Without algorithm a default (hmac-sha256) algorithm is used. The secret is a base64 encoded string. The changes are only in-memory and are gone next restart, for lasting changes edit the nsd.conf file or a file included from it.

assoc_tsig <zone> <key_name>
Associate the zone with the given tsig. The access control lists for notify, allow-notify, provide-xfr and request-xfr are adjusted to use the given key.

del_tsig <key_name>
Delete the TSIG key with the given name. Prints error if the key is still in use by some zone. The changes are only in-memory and are gone next restart, for lasting changes edit the nsd.conf file or a file included from it.

add_cookie_secret <secret>
Add or replace a cookie secret persistently. <secret> needs to be an 128 bit hex string.

Cookie secrets can be either active or staging. Active cookie secrets are used to create DNS Cookies, but verification of a DNS Cookie succeeds with any of the active or staging cookie secrets. The state of the current cookie secrets can be printed with the `print_cookie_secrets` command.

When there are no cookie secrets configured yet, the <secret> is added as active. If there is already an active cookie secret, the <secret> is added as staging or replacing an existing staging secret.

To "roll" a cookie secret used in an anycast set. The new secret has to be added as staging secret to **all** nodes in the anycast set. When all nodes can verify DNS Cookies with the new secret, the new secret can be activated with the `activate_cookie_secret` command. After all nodes have the new secret active for at least one hour, the previous secret can be dropped with the `drop_cookie_secret` command.

Persistence is accomplished by writing to a file which if configured with the **cookie-secret-file** option in the server section of the config file. The default value for that is: `/etc/nsd/nsd_cookiesecrets.txt`.

drop_cookie_secret
Drop the staging cookie secret.

activate_cookie_secret
Make the current staging cookie secret active, and the current active cookie secret staging.

print_cookie_secrets
Show the current configured cookie secrets with their status.

14.4 Exit Code

The `nsd-control` program exits with status code 1 on error, 0 on success.

14.5 Set Up

The setup requires a self-signed certificate and private keys for both the server and client. The script `nsd-control-setup` generates these in the default run directory, or with `-d` in another directory. If you change the access control permissions on the key files you can decide who can use `nsd-control`, by default owner and group but not all users. The script preserves private keys present in the directory. After running the script as root, turn on **control-enable** in *nsd.conf*.

14.6 Statistics Counters

The stats command shows a number of statistic counters.

num.queries
number of queries received (the tls, tcp and udp queries added up).

serverX.queries
number of queries handled by the server process. The number of server processes is set with the config statement **server-count**.

time.boot
uptime in seconds since the server was started. With fractional seconds.

time.elapsed
time since the last stats report, in seconds. With fractional seconds. Can be zero if polled quickly and the previous stats command resets the counters, so that the next gets a fully zero, and zero elapsed time, report.

size.db.disk
size of nsd.db on disk, in bytes.

size.db.mem
size of the DNS database in memory, in bytes.

size.xfrd.mem
size of memory for zone transfers and notifies in xfrd process, excludes TSIG data, in bytes.

size.config.disk
size of zonelist file on disk, excludes the nsd.conf size, in bytes.

size.config.mem
size of config data in memory, kept twice in server and xfrd process, in bytes.

num.type.X
number of queries with this query type.

num.opcode.X
number of queries with this opcode.

num.class.X
number of queries with this query class.

num.rcode.X
number of answers that carried this return code.

num.edns
number of queries with EDNS OPT.

num.ednserr
number of queries which failed EDNS parse.

num.udp
number of queries over UDP ip4.

num.udp6
number of queries over UDP ip6.

num.tcp
number of connections over TCP ip4.

num.tcp6
number of connections over TCP ip6.

num.tls
number of connections over TLS ip4. TLS queries are not part of num.tcp.

num.tls6
number of connections over TLS ip6. TLS queries are not part of num.tcp6.

num.answer_wo_aa
number of answers with NOERROR rcode and without AA flag, this includes the referrals.

num.rxerr
number of queries for which the receive failed.

num.txerr
number of answers for which the transmit failed.

num.raxfr
number of AXFR requests from clients (that got served with reply).

num.rixfr
number of IXFR requests from clients (that got served with reply).

num.truncated
number of answers with TC flag set.

num.dropped
number of queries that were dropped because they failed sanity check.

zone.master
number of master zones served. These are zones with no 'request-xfr:' entries.

zone.slave
number of slave zones served. These are zones with 'request-xfr' entries.

14.7 Files

/etc/nsd/nsd.conf
nsd configuration file.

/etc/nsd
directory with private keys (nsd_server.key and nsd_control.key) and self-signed certificates (nsd_server.pem and nsd_control.pem).

14.8 See Also

nsd.conf(5), *nsd(8)*, *nsd-checkconf(8)*

CHAPTER

FIFTEEN

CONFIGURE OPTIONS

NSD can be configured using GNU autoconf's configure script. In addition to standard configure options, one may use the following:

CC=compiler
Specify the C compiler. The default is gcc or cc. The compiler must support ANSI C89.

CPPFLAGS=flags
Specify the C preprocessor flags. Such as `-I<includedir>`.

CFLAGS=flags
Specify the C compiler flags. These include code generation, optimisation, warning, and debugging flags. These flags are also passed to the linker.

The default for gcc is `-g -O2`.

LD=linker
Specify the linker (defaults to the C compiler).

LDFLAGS=flags
Specify linker flags.

LIBS=libs
Specify additional libraries to link with.

--enable-root-server Configure NSD as a root server. Unless this option is specified, NSD will refuse to serve the `.` zone as a misconfiguration safeguard.

--disable-ipv6 Disables IPv6 support in NSD.

--enable-checking Enable some internal development checks. Useful if you want to modify NSD. This option enables the standard C "assert" macro and compiler warnings.

This will instruct NSD to be stricter when validating its input. This could lead to a reduced service level.

--enable-bind8-stats Enables BIND8-like statistics.

--enable-ratelimit Enables rate limiting, based on query name, type and source.

--enable-draft-rrtypes Enables draft RRtypes.

--with-configdir=dir Specified, NSD configuration directory, default `/etc/nsd`.

--with-nsd_conf_file=path Pathname to the NSD configuration file, default `/etc/nsd/nsd.conf`.

--with-pidfile=path Pathname to the NSD pidfile, default is platform specific, mostly `/var/run/nsd.pid`.

--with-dbfile=path Pathname to the NSD database, default is `/etc/nsd/nsd.db`.

--with-zonesdir=dir NSD default location for master zone files, default `/etc/nsd/`.

--with-user=username User name or ID to answer the queries with, default is `nsd`.

--with-facility=facility Specify the syslog facility to use. The default is LOG_DAEMON. See the syslog(3) manual page for the available facilities.

--with-libevent=path Specity the location of the `libevent` library (or libev). `--with-libevent=no` uses a builtin portable implementation (select()).

--with-ssl=path Specify the location of the OpenSSL libraries. OpenSSL 0.9.7 or higher is required for TSIG support.

--with-start_priority=number Startup priority for NSD.

--with-kill_priority=number Shutdown priority for NSD.

--with-tcp-timeout=number Set the default TCP timeout (in seconds). The default is 120 seconds.

--disable-nsec3 Disable NSEC3 support. With NSEC3 support enabled, very large zones, also non-NSEC3 zones, use about 20% more memory.

--disable-minimal-responses Disable minimal responses. If disabled, responses are more likely to get truncated, resulting in TCP fallback. When enabled (by default) NSD will leave out RRsets to make responses fit inside one datagram, but for shorter responses the full normal response is carried.

--disable-largefile Disable large file support (64 bit file lengths). Makes off_t a 32bit length during compilation.

CHAPTER

SIXTEEN

DIAGNOSING NSD LOG ENTRIES

NSD will print log messages to the system log (or `logfile:` configuration entry). Some of these messages are covered here.

Reload process <pid> failed with status <s>, continuing with old database
This log message indicates the reload process of NSD has failed for some reason. This can be anything from a missing database file to internal errors.

snipping off trailing partial part of <ixfr.db>
The file `ixfr.db` contains only part of expected data. The corruption is removed by snipping off the trailing part.

memory recyclebin holds <num> bytes
This is printed for every reload. NSD allocates and deallocates memory to service IXFR updates. The recycle bin holds deallocated memory ready for future use. If the number grows too large, a restart resets it.

xfrd: max number of tcp connections (32) reached
This line is printed when more than 32 zones need a zone transfer at the same time. The value is a compile constant (`xfrd-tcp.h`), but if this happens often for you, we could make this a config option. NSD will reuse existing TCP connections to the same primary (determined by IP address) to transfer up to 64k zones from that primary. Thus this error should only happen with more than 32 primaries or more than 64*32=2M zones that need to be updated at the same time.

If this happens, more zones have to wait until a zone transfer completes (or is aborted) before they can have a zone transfer too. This waiting list has no size limit.

error: <zone> NSEC3PARAM entry <num> has unknown hash algo <number>
This error means that the zone has NSEC3 chain(s) with hash algorithms that are not supported by this version of NSD, and thus cannot be served by NSD. If there are also no NSECs or NSEC3 chain(s) with known hash algorithms, NSD will not be able to serve DNSSEC authenticated denials for the zone.

CHAPTER

SEVENTEEN

GRAMMAR FOR DNS ZONE FILES

Note: It is near impossible to write a clean lexer/grammar for DNS (**RFC 1035**) zone files. At first it looks like it is easy to make such a beast, but when you start implementing it the details make it messy.

Since as early as NSD 1.4, the parser relies on Bison and Flex, tools for building programs that handle structured input. Compared to the previous implementation there is a slight decrease in speed (10-20%), but as the zone compiler is not critical to the performance of NSD, this not too relevant. The lexer part is located in the file zlexer.lex, the grammar is in zparser.y.

17.1 Zone File Lexer

Finding a good grammar and lexer for BIND zone files is rather hard. There are no real keywords and the meaning of most of the strings depends on the position relative to the other strings. An example, the following is a valid SOA record:

```
$ORIGIN example.org.
    SOA    soa    soa    ( 1 2 3 4 5 6 )
```

This SOA records means the administrator has an email address of `soa@example.org`. and the first nameserver is named `soa.example.org`. Both completely valid. The numbers are of course totally bogus.

Another example would be:

```
$ORIGIN example.org.
    SOA    soa    soa    ( 1 2 ) ( 3 4 ) ( 5 ) ( 6 )
```

The parsing of parentheses was also not trivial. Whitespace is also significant in zonefiles. The TAB before SOA has to be returned as previous_domain token by the lexer. Newlines inside parentheses are returned as SPACE which works but required some changes in the definitions of the resource records.

As shown above a simple `grep -i` for SOA does not do the trick. The lexer takes care of this tricky part by using an extra variable `in_rr` which is an enum containing: `outside`, `expecting_dname`, `after_dname`, `reading_type`. The semantics are as follows:

- `outside`, not in an RR (start of a line or a $-directive);
- `expecting_dname`, parse owner name of RR;
- `after_dname`, parse ttl, class;
- `reading_type`, we expect the RR type now;

With `in_rr` the lexer can say that in the first example above the first SOA is the actual record type, because it is located after a TAB. After we have found the TAB we set `in_rr` to `after_dname` which means we actually are expecting a RR type.

Again this is also not trivial because the class (IN) and TTL are also optional, if there are not specified we should substitute the current defaults from the zone we are parsing (this happens in the grammar). A DNS zone file is further complicated by the unknown RR record types.

17.2 Zone File Grammar

After the lexer was written the grammar itself is quite clean and nice. The basic idea is that every RR consists of single line (the parentheses are handled in the lexer - so this really is the case). If a line is not a RR it is either a comment, empty or a $-directive. Some $-directives are handled inside the lexer ($INCLUDE) while others ($ORIGIN) must be dealt with inside the grammer.

An RR is defined as:

```
rr:     ORIGIN SP rrrest
```

and:

```
rrrset: classttl rtype
```

And then we have a whole list of:

```
rtype: TXT sp rdata_txt
       | DS sp rdata_ds
       | AAAA sp rdata_aaaa
```

which are then parsed by using the `rdata_` rule. Shown here is the one for the SOA:

```
rdata_soa:  dname sp dname sp STR sp STR sp STR sp STR sp STR trail
    {
        /* convert the soa data */
        zadd_rdata_domain( current_parser, $1); /* prim. ns */
        zadd_rdata_domain( current_parser, $3); /* email */
        zadd_rdata_wireformat( current_parser,  \
                zparser_conv_rdata_period(zone_region, $5.str) ); /* serial */
        zadd_rdata_wireformat( current_parser,  \
                zparser_conv_rdata_period(zone_region, $7.str) ); /* refresh */
        zadd_rdata_wireformat( current_parser,  \
                zparser_conv_rdata_period(zone_region, $9.str) ); /* retry */
        zadd_rdata_wireformat( current_parser,  \
                zparser_conv_rdata_period(zone_region, $11.str) ); /* expire */
        zadd_rdata_wireformat( current_parser,  \
                zparser_conv_rdata_period(zone_region, $13.str) ); /* minimum */

        /* XXX also store the minium in case of no TTL? */
        if ( (current_parser->minimum = zparser_ttl2int($11.str) ) == -1 )
            current_parser->minimum = DEFAULT_TTL;
    };
```

The semantic actions in the grammer store the RR data for processing by the zone compiler. The resulting database is then used by NSD the serve the data.

INDEX