Knot DNS Documentation

Release 2.5.7

2018-01-02

CHAPTER

ONE

INTRODUCTION

1.1 What is Knot DNS

Knot DNS is a high-performance open-source DNS server. It implements only the authoritative domain name service. Knot DNS can reliably serve TLD domains as well as any other zones.

Knot DNS benefits from its multi-threaded and mostly lock-free implementation which allows it to scale well on SMP systems and operate non-stop even when adding or removing zones.

For more info and downloads see www.knot-dns.cz.

1.2 Knot DNS features

DNS features:

- Master and slave operation
- Internet class (IN)
- DNS extension (EDNS0)
- TCP and UDP protocols
- Dynamic zone updates
- DNSSEC with NSEC and NSEC3
- Transaction signature using TSIG
- Full and incremental zone transfers (AXFR, IXFR)
- Name server identification using NSID or Chaos TXT records
- Resource record types A, NS, CNAME, SOA, PTR, HINFO, MINFO, MX, TXT, RP, AFSDB, RT, KEY, AAAA, LOC, SRV, NAPTR, KX, CERT, DNAME, APL, DS, SSHFP, IPSECKEY, RRSIG, NSEC, DNSKEY, DHCID, NSEC3, NSEC3PARAM, TLSA, CDS, CDNSKEY, SPF, NID, L32, L64, LP, EUI48, EUI64, URI, CAA, and Unknown

Server features:

- IPv4 and IPv6 support
- Semantic zone checks
- Server control interface
- Zone journal storage
- Persistent zone event timers
- YAML-based or database-based configuration
- Query processing modules with dynamic loading

- On-the-fly zone management and server reconfiguration
- Automatic DNSSEC signing with automatic key maganement and PKCS #11 interface

Remarkable module extensions:

- Response rate limiting
- Forward and reverse records synthesis
- DNS request traffic statistics
- Dnstap traffic logging
- Online DNSSEC signing

1.3 License

Knot DNS is licensed under the GNU General Public License version 3 or (at your option) any later version. The full text of the license is available in the `COPYING` file distributed with source code.

CHAPTER

TWO

REQUIREMENTS

2.1 Hardware

Knot DNS requirements are not very demanding for typical installations, and a commodity server or a virtual solution will be sufficient in most cases.

However, please note that there are some scenarios that will require administrator's attention and some testing of exact requirements before deploying Knot DNS to a production environment. These cases include deployment for a large number of zones (DNS hosting), large number of records in one or more zones (TLD), or large number of requests.

2.1.1 CPU requirements

The server scales with processing power and also with the number of available cores/CPUs. Enabling Hyper-threading is convenient if supported.

There is no lower bound on the CPU requirements, but it should support memory barriers and CAS (i586 and newer).

2.1.2 Network card

The best results have been achieved with multi-queue network cards. The number of multi-queues should equal the total number of CPU cores (with Hyper-threading enabled).

2.1.3 Memory requirements

The server implementation focuses on performance and thus can be quite memory demanding. The rough estimate for memory requirements is 3 times the size of the zone in the text format. Again this is only an estimate and you are advised to do your own measurements before deploying Knot DNS to production.

Note: To ensure uninterrupted serving of the zone, Knot DNS employs the Read-Copy-Update mechanism instead of locking and thus requires twice the amount of memory for the duration of incoming transfers.

2.2 Operating system

Knot DNS itself is written in a portable way and can be compiled and run on most UNIX-like systems, such as Linux, *BSD, and macOS.

2.3 Required libraries

Knot DNS requires a few libraries to be available:

- libedit
- GnuTLS >= 3.3
- Userspace RCU >= 0.5.4
- lmdb >= 0.9.15

Note: The LMDB library is included with the Knot DNS source code, however linking with the system library is preferred.

2.4 Optional libraries

International Domain Names support (IDNA2003 or IDNA2008) in kdig:

- libidn or libidn2

Systemd's startup notifications mechanism and journald logging:

- libsystemd

Dnstap support in kdig and module dnstap:

- fstrm (and protobuf-c if building from the source code)

POSIX 1003.1e *capabilites(7)* by sandboxing the exposed threads. Most rights are stripped from the exposed threads for security reasons.

- libcap-ng >= 0.6.4

CHAPTER

THREE

INSTALLATION

3.1 Installation from a package

Knot DNS may already be included in your operating system distribution and therefore can be installed from packages (Linux), ports (BSD), or via Homebrew (macOS). This is always preferred unless you want to test the latest features, contribute to Knot development, or you just know what you are doing.

See the project download page for the latest information.

3.2 Installation from the source code

3.2.1 Required build environment

The build process relies on these standard tools:

- make
- libtool
- pkg-config
- autoconf >= 2.65
- python-sphinx (optional, for documentation building)

GCC at least 4.1 is strictly required for atomic built-ins, but the latest available version is recommended. Another requirements `_GNU_SOURCE` and C99 support, otherwise it adapts to the compiler available features. LLVM clang compiler since version 2.9 can be used as well.

3.2.2 Getting the source code

You can find the source code for the latest release on www.knot-dns.cz. Alternatively, you can fetch the whole project from the git repository git://git.nic.cz/knot-dns.git.

After obtaining the source code, the compilation and installation is a quite straightforward process using autotools.

3.2.3 Configuring and generating Makefiles

If compiling from the git source, you need to bootstrap the `./configure` file first:

```
$ autoreconf -i -f
```

In most cases, you can just run configure without any options:

```
$ ./configure
```

For all available configure options run:

```
$ ./configure --help
```

3.2.4 Compilation

After running `./configure` you can compile Knot DNS by running `make` command, which will produce binaries and other related files:

```
$ make
```

Note: The compilation with enabled optimizations may take a long time. In such a case the `--disable-fastparser` configure option can help.

3.2.5 Installation

When you have finished building Knot DNS, it's time to install the binaries and configuration files into the operation system hierarchy. You can do so by executing:

```
$ make install
```

When installing as a non-root user, you might have to gain elevated privileges by switching to root user, e.g. `sudo make install` or `su -c 'make install'`.

CHAPTER

FOUR

CONFIGURATION

4.1 Simple configuration

The following example presents a simple configuration file which can be used as a base for your Knot DNS setup:

```
# Example of a very simple Knot DNS configuration.

server:
    listen: 0.0.0.0@53
    listen: ::@53

zone:
  - domain: example.com
    storage: /var/lib/knot/zones/
    file: example.com.zone

log:
  - target: syslog
    any: info
```

Now let's walk through this configuration step by step:

- The *listen* statement in the *server section* defines where the server will listen for incoming connections. We have defined the server to listen on all available IPv4 and IPv6 addresses, all on port 53.
- The *zone section* defines the zones that the server will serve. In this case, we defined one zone named *example.com* which is stored in the zone file `/var/lib/knot/zones/example.com.zone`.
- The *log section* defines the log facilities for the server. In this example, we told Knot DNS to send its log messages with the severity `info` or more serious to the syslog.

For detailed description of all configuration items see *Configuration Reference*.

4.2 Zone templates

A zone template allows a single zone configuration to be shared among several zones. The `default` template identifier is reserved for the default template:

```
template:
  - id: default
    storage: /var/lib/knot/master
    semantic-checks: on

  - id: signed
    storage: /var/lib/knot/signed
    dnssec-signing: on
    semantic-checks: on
    master: [master1, master2]
```

```
  - id: slave
    storage: /var/lib/knot/slave

zone:
  - domain: example1.com     # Uses default template

  - domain: example2.com     # Uses default template
    semantic-checks: off     # Override default settings

  - domain: example.cz
    template: signed
    master: master3          # Override masters to just master3

  - domain: example1.eu
    template: slave
    master: master1

  - domain: example2.eu
    template: slave
    master: master2
```

Note: Each template option can be explicitly overridden in zone-specific configuration.

4.3 Access control list (ACL)

An ACL list specifies which remotes are allowed to send the server a specific request. A remote can be a single IP address or a network subnet. Also a TSIG key can be assigned (see *keymgr* how to generate a TSIG key).

With no ACL rule, all the actions are denied for the zone. Each ACL rule can allow one or more actions for given address/subnet/TSIG, or deny them.

The rule precendence, if multiple rules match (e.g. overlapping address ranges), is not for stricter or more specific rules. In any case, just the first – in the order of rules in zone or template acl configuration item, not in the order of declarations in acl section – matching rule applies and the rest is ignored.

See following examples and *ACL section*.:

```
acl:
  - id: address_rule
    address: [2001:db8::1, 192.168.2.0/24]
    action: transfer

  - id: deny_rule
    address: 192.168.2.100
    action: transfer
    deny: on

zone:
  - domain: acl1.example.com.
    acl: [deny_rule, address_rule] # deny_rule first here to take precendence
```

```
key:
  - id: key1                  # The real TSIG key name
    algorithm: hmac-md5
    secret: Wg==

acl:
```

```
  - id: deny_all
    address: 192.168.3.0/24
    deny: on # no action specified and deny on implies denial of all actions

  - id: key_rule
    key: key1                  # Access based just on TSIG key
    action: [transfer, notify]

zone:
  - domain: acl2.example.com
    acl: [deny_all, key_rule]
```

4.4 Slave zone

Knot DNS doesn't strictly differ between master and slave zones. The only requirement is to have a *master* statement set for the given zone. Also note that you need to explicitly allow incoming zone changed notifications via `notify` *action* through zone's *acl* list, otherwise the update will be rejected by the server. If the zone file doesn't exist it will be bootstrapped over AXFR:

```
remote:
  - id: master
    address: 192.168.1.1@53

acl:
  - id: notify_from_master
    address: 192.168.1.1
    action: notify

zone:
  - domain: example.com
    storage: /var/lib/knot/zones/
    # file: example.com.zone   # Default value
    master: master
    acl: notify_from_master
```

Note that the *master* option accepts a list of multiple remotes. The remotes should be listed according to their preference. The first remote has the highest preference, the other remotes are used for failover. When the server receives a zone update notification from a listed remote, that remote will be the most preferred one for the subsequent transfer.

To use TSIG for transfers and notification messages authentication, configure a TSIG key and assign the key both to the remote and the ACL rule. Notice that the *remote* and *ACL* definitions are independent:

```
key:
  - id: slave1_key
    algorithm: hmac-md5
    secret: Wg==

remote:
  - id: master
    address: 192.168.1.1@53
    key: slave1_key

acl:
  - id: notify_from_master
    address: 192.168.1.1
    key: slave1_key
    action: notify
```

Note: When transferring a lot of zones, the server may easily get into a state when all available ports are in the TIME_WAIT state, thus the transfers seize until the operating system closes the ports for good. There are several ways to work around this:

- Allow reusing of ports in TIME_WAIT (sysctl -w net.ipv4.tcp_tw_reuse=1)
- Shorten TIME_WAIT timeout (tcp_fin_timeout)
- Increase available local port count

4.5 Master zone

An ACL with the `transfer` action must be configured to allow outgoing zone transfers. An ACL rule consists of a single address or a network subnet:

```
remote:
  - id: slave1
    address: 192.168.2.1@53

acl:
  - id: slave1_acl
    address: 192.168.2.1
    action: transfer

  - id: others_acl
    address: 192.168.3.0/24
    action: transfer

zone:
  - domain: example.com
    storage: /var/lib/knot/zones/
    file: example.com.zone
    notify: slave1
    acl: [slave1_acl, others_acl]
```

Optionally, a TSIG key can be specified:

```
key:
  - id: slave1_key
    algorithm: hmac-md5
    secret: Wg==

remote:
  - id: slave1
    address: 192.168.2.1@53
    key: slave1_key

acl:
  - id: slave1_acl
    address: 192.168.2.1
    key: slave1_key
    action: transfer

  - id: others_acl
    address: 192.168.3.0/24
    action: transfer
```

Note that a slave zone may serve as a master zone at the same time:

```
remote:
  - id: master
    address: 192.168.1.1@53
  - id: slave1
    address: 192.168.2.1@53

acl:
  - id: notify_from_master
    address: 192.168.1.1
    action: notify

  - id: slave1_acl
    address: 192.168.2.1
    action: transfer

  - id: others_acl
    address: 192.168.3.0/24
    action: transfer

zone:
  - domain: example.com
    storage: /var/lib/knot/zones/
    file: example.com.zone
    master: master
    notify: slave1
    acl: [notify_from_master, slave1_acl, others_acl]
```

4.6 Dynamic updates

Dynamic updates for the zone are allowed via proper ACL rule with the `update` action. If the zone is configured as a slave and a DNS update message is accepted, the server forwards the message to its primary master. The master's response is then forwarded back to the originator.

However, if the zone is configured as a master, the update is accepted and processed:

```
acl:
  - id: update_acl
    address: 192.168.3.0/24
    action: update

zone:
  - domain: example.com
    file: example.com.zone
    acl: update_acl
```

4.7 Automatic DNSSEC signing

Knot DNS supports automatic DNSSEC signing for static zones. The signing can operate in two modes:

1. *Automatic key management*. In this mode, the server maintains signing keys. New keys are generated according to assigned policy and are rolled automatically in a safe manner. No zone operator intervention is necessary.
2. *Manual key management*. In this mode, the server maintains zone signatures only. The signatures are kept up-to-date and signing keys are rolled according to timing parameters assigned to the keys. The keys must be generated and timing parameters must be assigned by the zone operator.

The DNSSEC signing process maintains some metadata which is stored in the KASP (Key And Signature Policy) database. This database is backed by LMDB.

Warning: Make sure to set the KASP database permissions correctly. For manual key management, the database must be *readable* by the server process. For automatic key management, it must be *writeable*. If no HSM is used, the database also contains private key material – don't set the permissions too weak.

4.7.1 Automatic ZSK management

For automatic ZSK management, a signing policy has to be configured and assigned to the zone. The policy specifies how the zone is signed (i.e. signing algorithm, key size, key lifetime, signature lifetime, etc.). The policy can be configured in the *policy section*, or a `default` policy with the default parameters can be used.

A minimal zone configuration may look as follows:

```
zone:
  - domain: myzone.test
    dnssec-signing: on
    dnssec-policy: default
```

With custom signing policy, the policy section will be added:

```
policy:
  - id: rsa
    algorithm: RSASHA256
    ksk-size: 2048
    zsk-size: 1024

zone:
  - domain: myzone.test
    dnssec-signing: on
    dnssec-policy: rsa
```

After configuring the server, reload the changes:

```
$ knotc reload
```

The server will generate initial signing keys and sign the zone properly. Check the server logs to see whether everything went well.

Warning: This guide assumes that the zone *myzone.test* was not signed prior to enabling the automatic key management. If the zone was already signed, all existing keys must be imported using `keymgr import-bind` command before enabling the automatic signing. Also the algorithm in the policy must match the algorithm of all imported keys. Otherwise the zone will be resigned at all.

4.7.2 Automatic KSK management

For automatic KSK management, first configure ZSK management like above, and use additional options in *policy section*, mostly specifying desired (finite) lifetime for KSK:

```
remote:
  - id: test_zone_server
    address: 192.168.12.1@53

submission:
  - id: test_zone_sbm
```

```
    parent: [test_zone_server]

policy:
  - id: rsa
    algorithm: RSASHA256
    ksk-size: 2048
    zsk-size: 1024
    zsk-lifetime: 30d
    ksk-lifetime: 365d
    ksk-submission: test_zone_sbm

zone:
  - domain: myzone.test
    dnssec-signing: on
    dnssec-policy: rsa
```

After the initially-generated KSK reaches its lifetime, new KSK is published and after convenience delay the submission is started. The server publishes CDS and CDNSKEY records and the user shall propagate them to the parent. The server periodically checks for DS at the master and when positive, finishes the rollover.

To share KSKs among zones, set the ksk-shared policy parameter. It is strongly discouraged to change the policy `id` afterwards! The shared key's creation timestamp will be equal for all zones, but other timers (e.g. activate, retire) may get out of sync.

```
policy:
  - id: shared
    ...
    ksk-shared: true

zone:
  - domain: firstzone.test
    dnssec-signing: on
    dnssec-policy: shared

zone:
  - domain: secondzone.test
    dnssec-signing: on
    dnssec-policy: shared
```

4.7.3 Manual key management

For automatic DNSSEC signing with manual key management, a signing policy with manual key management flag has to be set:

```
policy:
  - id: manual
    manual: on

zone:
  - domain: myzone.test
    dnssec-signing: on
    dnssec-policy: manual
```

To generate signing keys, use the *keymgr* utility. Let's use the Single-Type Signing scheme with two algorithms. Run:

```
$ keymgr myzone.test. generate algorithm=RSASHA256 size=1024
$ keymgr myzone.test. generate algorithm=ECDSAP256SHA256 size=256
```

And reload the server. The zone will be signed.

To perform a manual rollover of a key, the timing parameters of the key need to be set. Let's roll the RSA key. Generate a new RSA key, but do not activate it yet:

```
$ keymgr myzone.test. generate algorithm=RSASHA256 size=1024 active=now+1d
```

Take the key ID (or key tag) of the old RSA key and disable it the same time the new key gets activated:

```
$ keymgr myzone.test. set <old_key_id> retire=now+1d remove=now+1d
```

Reload the server again. The new key will be published (i.e. the DNSKEY record will be added into the zone). Do not forget to update the DS record in the parent zone to include a reference to the new RSA key. This must happen in one day (in this case) including a delay required to propagate the new DS to caches.

Note that as the `+1d` time specification is computed from the current time, the key replacement will not happen at once. First, a new key will be activated. A few moments later, the old key will be deactivated and removed. You can use exact time specification to make these two actions happen in one go.

4.7.4 Zone signing

The signing process consists of the following steps:

1. Processing KASP database events. (e.g. performing a step of a rollover).
2. Updating the DNSKEY records. The whole DNSKEY set in zone apex is replaced by the keys from the KASP database. Note that keys added into the zone file manually will be removed. To add an extra DNSKEY record into the set, the key must be imported into the KASP database (possibly deactivated).
3. Fixing the NSEC or NSEC3 chain.
4. Removing expired signatures, invalid signatures, signatures expiring in a short time, and signatures issued by an unknown key.
5. Creating missing signatures. Unless the Single-Type Signing Scheme is used, DNSKEY records in a zone apex are signed by KSK keys and all other records are signed by ZSK keys.
6. Updating and resigning SOA record.

The signing is initiated on the following occasions:

- Start of the server
- Zone reload
- Reaching the signature refresh period
- Key set changed due to rollover event
- Received DDNS update
- Forced zone resign via server control interface

On a forced zone resign, all signatures in the zone are dropped and recreated.

The `knotc zone-status` command can be used to see when the next scheduled DNSSEC resign will happen.

4.7.5 Limitations

The current DNSSEC implementation in Knot DNS has some limitations. Most of the limitations will be hopefully removed in the near future.

- Automatic key management:
  - Only one DNSSEC algorithm can be used per zone.
  - ZSK rollover always uses key pre-publish method (actually a feature).
  - KSK rollover always uses pre-publish double-ksk method.

- Signing:
  - Signature expiration jitter is not implemented.
  - Signature expiration skew is not implemented.
- Utilities:
  - Legacy key import requires a private key.
  - Legacy key export is not implemented.

4.8 Query modules

Knot DNS supports configurable query modules that can alter the way queries are processed. Each query requires a finite number of steps to be resolved. We call this set of steps a *query plan*, an abstraction that groups these steps into several stages.

- Before-query processing
- Answer, Authority, Additional records packet sections processing
- After-query processing

For example, processing an Internet-class query needs to find an answer. Then based on the previous state, it may also append an authority SOA or provide additional records. Each of these actions represents a 'processing step'. Now, if a query module is loaded for a zone, it is provided with an implicit query plan which can be extended by the module or even changed altogether.

A module is active if its name, which includes the `mod-` prefix, is assigned to the zone/template *module* option or to the *default* template *global-module* option if activating for all queries. If the module is configurable, a corresponding module section with an identifier must be created and then referenced in the form of `module_name/module_id`. See *Modules* for the list of available modules.

Note: Query modules are processed in the order they are specified in the zone/template configuration. In most cases, the recommended order is:

```
mod-synthrecord, mod-onlinesign, mod-rrl, mod-dnstap, mod-stats
```

4.9 Performance Tuning

4.9.1 Numbers of Workers

There are three types of workers ready for parallel execution of performance-oriented tasks: UDP workers, TCP workers, and Background workers. The first two types handle all network requests coming through UDP and TCP protocol (respectively) and do all the response job for common queries. Background workers process changes to the zone.

By default, Knot determines well-fitting number of workers based on the number of CPU cores. The user can specify the numbers of workers for each type with configuration/server section: *udp-workers*, *tcp-workers*, *background-workers*.

An indication on when to increase number of workers is a situation when the server is lagging behind the expected performance, while the CPU usage is low. This is usually because of waiting for network or I/O response during the operation. It may be caused by Knot design not fitting well the usecase. The user should try increasing the number of workers (of the related type) slightly above 100 and if the performance gets better, he can decide about further exact setting.

4.9.2 Sysctl and NIC optimizations

There are several recommendations based on Knot developers' experience with their specific HW and SW (mainstream Intel-based servers, Debian-based GNU/Linux distribution). They may or may not positively (or negatively) influence performance in common use cases.

If your NIC driver allows it (see /proc/interrupts for hint), set CPU affinity (/proc/irq/$IRQ/smp_affinity) manually so that each NIC channel is served by unique CPU core(s). You must turn off irqbalance service before to avoid configuration override.

Configure sysctl as follows:

```
socket_bufsize=1048576
busy_latency=0
backlog=40000
optmem_max=20480

net.core.wmem_max     = $socket_bufsize
net.core.wmem_default = $socket_bufsize
net.core.rmem_max     = $socket_bufsize
net.core.rmem_default = $socket_bufsize
net.core.busy_read = $busy_latency
net.core.busy_poll = $busy_latency
net.core.netdev_max_backlog = $backlog
net.core.optmem_max = $optmem_max
```

Disable huge pages.

Configure your CPU to "performance" mode. This can be achieved depending on architecture, e.g. in BIOS, or e.g. configuring /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor to "performance".

Tune your NIC device with ethtool:

```
ethtool -A $dev autoneg off rx off tx off
ethtool -K $dev tso off gro off ufo off
ethtool -G $dev rx 4096 tx 4096
ethtool -C $dev rx-usecs 75
ethtool -C $dev tx-usecs 75
ethtool -N $dev rx-flow-hash udp4 sdfn
ethtool -N $dev rx-flow-hash udp6 sdfn
```

On FreeBSD you can just:

```
ifconfig ${dev} -rxcsum -txcsum -lro -tso
```

Knot developers are open to hear about users' further suggestions about network devices tuning/optimization.

CHAPTER

FIVE

OPERATION

The Knot DNS server part `knotd` can run either in the foreground, or in the background using the `-d` option. When run in the foreground, it doesn't create a PID file. Other than that, there are no differences and you can control both the same way.

The tool `knotc` is designed as a user front-end, making it easier to control running server daemon. If you want to control the daemon directly, use `SIGINT` to quit the process or `SIGHUP` to reload the configuration.

If you pass neither configuration file (`-c` parameter) nor configuration database (`-C` parameter), the server will first attempt to use the default configuration database stored in `/var/lib/knot/confdb` or the default configuration file stored in `/etc/knot/knot.conf`. Both the default paths can be reconfigured with `--with-storage=path` or `--with-configdir=path` respectively.

Example of server start as a daemon:

```
$ knotd -d -c knot.conf
```

Example of server shutdown:

```
$ knotc -c knot.conf stop
```

For a complete list of actions refer to the program help (`-h` parameter) or to the corresponding manual page.

Also, the server needs to create *rundir* and *storage* directories in order to run properly.

5.1 Configuration database

In the case of a huge configuration file, the configuration can be stored in a binary database. Such a database can be simply initialized:

```
$ knotc conf-init
```

or preloaded from a file:

```
$ knotc conf-import input.conf
```

Also the configuration database can be exported into a textual file:

```
$ knotc conf-export output.conf
```

Warning: The import and export commands access the configuration database directly, without any interaction with the server. So it is strictly recommended to perform these operations when the server is not running.

5.2 Dynamic configuration

The configuration database can be accessed using the server control interface during the running server. To get the full power of the dynamic configuration, the server must be started with a specified configuration database location or with the default database initialized. Otherwise all the changes to the configuration will be temporary (until the server stop).

Note: The database can be *imported* in advance.

Most of the commands get an item name and value parameters. The item name is in the form of `section[identifier].name`. If the item is multivalued, more values can be specified as individual (command line) arguments. Beware of the possibility of pathname expansion by the shell. For this reason, slashed square brackets or quoted parameters is advisable.

To get the list of configuration sections or to get the list of section items:

```
$ knotc conf-list
$ knotc conf-list 'server'
```

To get the whole configuration or to get the whole configuration section or to get all section identifiers or to get a specific configuration item:

```
$ knotc conf-read
$ knotc conf-read 'remote'
$ knotc conf-read 'zone.domain'
$ knotc conf-read 'zone[example.com].master'
```

Warning: The following operations don't work on OpenBSD!

Modifying operations require an active configuration database transaction. Just one transaction can be active at a time. Such a transaction then can be aborted or committed. A semantic check is executed automatically before every commit:

```
$ knotc conf-begin
$ knotc conf-abort
$ knotc conf-commit
```

To set a configuration item value or to add more values or to add a new section identifier or to add a value to all identified sections:

```
$ knotc conf-set 'server.identity' 'Knot DNS'
$ knotc conf-set 'server.listen' '0.0.0.0@53' '::@53'
$ knotc conf-set 'zone[example.com]'
$ knotc conf-set 'zone.slave' 'slave2'
```

Note: Also the include operation can be performed. A non-absolute file location is relative to the server binary path, not to the control binary path!:

```
$ knotc conf-set 'include' '/tmp/new_zones.conf'
```

To unset the whole configuration or to unset the whole configuration section or to unset an identified section or to unset an item or to unset a specific item value:

```
$ knotc conf-unset
$ knotc conf-unset 'zone'
```

```
$ knotc conf-unset 'zone[example.com]'
$ knotc conf-unset 'zone[example.com].master'
$ knotc conf-unset 'zone[example.com].master' 'remote2' 'remote5'
```

To get the change between the current configuration and the active transaction for the whole configuration or for a specific section or for a specific identified section or for a specific item:

```
$ knotc conf-diff
$ knotc conf-diff 'zone'
$ knotc conf-diff 'zone[example.com]'
$ knotc conf-diff 'zone[example.com].master'
```

An example of possible configuration initialization:

```
$ knotc conf-begin
$ knotc conf-set 'server.listen' '0.0.0.0@53' '::@53'
$ knotc conf-set 'remote[master_server]'
$ knotc conf-set 'remote[master_server].address' '192.168.1.1'
$ knotc conf-set 'template[default]'
$ knotc conf-set 'template[default].storage' '/var/lib/knot/zones/'
$ knotc conf-set 'template[default].master' 'master_server'
$ knotc conf-set 'zone[example.com]'
$ knotc conf-diff
$ knotc conf-commit
```

5.3 Slave mode

Running the server as a slave is very straightforward as you usually bootstrap zones over AXFR and thus avoid any manual zone operations. In contrast to AXFR, when the incremental transfer finishes, it stores the differences in the journal file and doesn't update the zone file immediately but after the *zonefile-sync* period elapses.

5.4 Master mode

If you just want to check the zone files before starting, you can use:

```
$ knotc zone-check example.com
```

For an approximate estimation of server's memory consumption, you can use:

```
$ knotc zone-memstats example.com
```

This action prints the count of resource records, percentage of signed records and finally estimation of memory consumption for each zone, unless specified otherwise. Please note that the estimated values may differ from the actual consumption. Also, for slave servers with incoming transfers enabled, be aware that the actual memory consumption might be double or higher during transfers.

5.5 Reading and editing zones

Knot DNS allows you to read or change zone contents online using server control interface.

Warning: Avoid concurrent zone file modification, and/or dynamic updates, and/or zone changing over control interface. Otherwise, the zone could be inconsistent.

To get contents of all configured zones, or a specific zone contents, or zone records with a specific owner, or even with a specific record type:

```
$ knotc zone-read --
$ knotc zone-read example.com
$ knotc zone-read example.com ns1
$ knotc zone-read example.com ns1 NS
```

Note: If the record owner is not a fully qualified domain name, then it is considered as a relative name to the zone name.

To start a writing transaction on all zones or on specific zones:

```
$ knotc zone-begin --
$ knotc zone-begin example.com example.net
```

Now you can list all nodes within the transaction using the `zone-get` command, which always returns current data with all changes included. The command has the same syntax as `zone-read`.

Within the transaction, you can add a record to a specific zone or to all zones with an open transaction:

```
$ knotc zone-set example.com ns1 3600 A 192.168.0.1
$ knotc zone-set -- ns1 3600 A 192.168.0.1
```

To remove all records with a specific owner, or a specific rrset, or a specific record data:

```
$ knotc zone-unset example.com ns1
$ knotc zone-unset example.com ns1 A
$ knotc zone-unset example.com ns1 A 192.168.0.2
```

To see the difference between the original zone and the current version:

```
$ knotc zone-diff example.com
```

Finally, either commit or abort your transaction:

```
$ knotc zone-commit example.com
$ knotc zone-abort example.com
```

A full example of setting up a completely new zone from scratch:

```
$ knotc conf-begin
$ knotc conf-set zone.domain example.com
$ knotc conf-commit
$ knotc zone-begin example.com
$ knotc zone-set example.com @ 7200 SOA ns hostmaster 1 86400 900 691200 3600
$ knotc zone-set example.com ns 3600 A 192.168.0.1
$ knotc zone-set example.com www 3600 A 192.168.0.100
$ knotc zone-commit example.com
```

5.6 Journal behaviour

Zone journal keeps some history of changes of the zone. It is useful for responding IXFR queries. Also if zone file flush is disabled, journal keeps diff between zonefile and zone for the case of server shutdown. The history is stored by changesets - diffs of zone contents between two (usually subsequent) zone serials.

Journals for all zones are stored in common LMDB database. Huge changesets are split into 70 KiB (this constant is hardcoded) blocks to prevent fragmentation of the DB. Journal does each operation in one transaction to keep

consistency of the DB and performance. The exception is when store transaction exceeds 5% of the whole DB mapsize, it is split into multiple ones and some dirty-chunks-management involves.

Each zone journal has own usage limit on how much DB space it may occupy. Before hitting the limit, changesets are stored one-by-one and whole history is linear. While hitting the limit, the zone is flushed into zone file, and oldest changesets are deleted as needed to free some space. Actually, twice (again, hardcoded constant) the needed amount is deleted to prevent too frequent deletes. Further zone file flush is invoked after the journal runs out of deletable "flushed changesets".

If zone file flush is disabled, instead of flushing the zone, the journal tries to save space by merging older changesets into one. It works well if the changes rewrite each other, e.g. periodically changing few zone records, re-signing whole zone... The diff between zone file and zone is thus preserved, even if journal deletes some older changesets.

5.7 DNSSEC keys algorithm rollover

Algorithm rollover is a process of changing DNSSEC signing keys, where the new keys are of different algorithm. The zone signatures must never go Bogus during the process, even considering records cached in resolvers. The process is generally described in RFC 6781. Following are some hints how to implement algorithm rollover when using Knot DNS.

The prerequisite is having a zone with automatic DNSSEC signing enabled, active KSK and ZSK present. (The CSK case should work analogously, not mentioned further.) It is recommended to disable automatic key management during the rollover. Note that from the view of common key rollovers, here we must put the keys into a weird state: active, but not published. This is done by hard-setting their timers so that active < publish < retire (whereas standard rollovers have publish < active < retire).

First we need to generate new keys. They must be first used for signing, and after some period (propagation delay let's say 1h + zone records' TTL let's say 1h) published. We have to preprate the timestamps carefully, using the notation 'now+2h' can be creepy with "now" changing between the Keymgr invokes. We then re-sign the zone just to force knotd to reload zone keys:

```
$ NOW=$(date +%s)
$ NOW2H=$((NOW + 7200))
$ keymgr example.com. generate algorithm=14 size=384 ksk=yes \
                               ready=$NOW2H active=0 publish=$NOW2H
$ keymgr example.com. generate algorithm=14 size=384 ksk=no \
                               ready=$NOW active=$NOW publish=$NOW2H
$ knotc zone-sign example.com.
```

After waiting for the keys to get published as scheduled, we may tell the parent zone operator to renew our DS record. As the KSK is in ready state, we have the CDS/CDNSKEY records in our zone. After waiting again for some propagation period, we continue with removing old KSK and putting old ZSK into active-not-published state (we must first obtain the keys' IDs with 'keymgr example.com. list'). We may also confirm the new KSK submission (which reloads KASP DB as a side-effect):

```
$ keymgr example.com. set $OLD_KSK_ID retire=now+0 remove=now+0
$ keymgr example.com. set $OLD_ZSK_ID publish=0
$ knotc zone-ksk-submitted example.com.
```

Finally, after one more propagation period, we remove old ZSK:

```
$ keymgr example.com. set $OLD_ZSK_ID retire=now+0 remove=now+0
$ knotc zone-sign example.com.
```

5.8 Daemon controls

Knot DNS was designed to allow server reconfiguration on-the-fly without interrupting its operation. Thus it is possible to change both configuration and zone files and also add or remove zones without restarting the server. This can be done with:

```
$ knotc reload
```

If you want to enable ixfr differences creation from changes you make to a zone file, enable *ixfr-from-differences* in the zone configuration and reload your server as seen above. If *SOA*'s *serial* is not changed, no differences will be created.

If you want to refresh the slave zones, you can do this with:

```
$ knotc zone-refresh
```

5.9 Statistics

The server provides some general statistics and optional query module statistics (see *mod-stats*).

Server statistics or global module statistics can be shown by:

```
$ knotc stats
$ knotc stats server              # Show all server counters
$ knotc stats mod-stats           # Show all mod-stats counters
$ knotc stats server.zone-count   # Show specific server counter
```

Per zone statistics can be shown by:

```
$ knotc zone-stats example.com mod-stats
```

To show all supported counters even with 0 value use the force option.

A simple periodic statistic dumping to a YAML file can also be enabled. See *Statistics section* for the configuration details.

As the statistics data can be accessed over the server control socket, it is possible to create an arbitrary script (Python is supported at the moment) which could, for example, publish the data in the JSON format via HTTP(S) or upload the data to a more efficient time series database. Take a look into the python folder of the project for these scripts.

CHAPTER

SIX

TROUBLESHOOTING

First of all, check the logs. Enabling at least the `warning` message severity may help you to identify some problems. See the *Logging section* for details.

6.1 Reporting bugs

If you are unable to solve the problem by yourself, you can submit a bugreport to the Knot DNS developers. For security or sensitive issues contact the developers directly on knot-dns@labs.nic.cz. All other bugs and questions may be directed to the public Knot DNS users mailing list (knot-dns-users@lists.nic.cz) or may be entered into the issue tracking system.

Before anything else, please try to answer the following questions:

- Has it been working?
- What has changed? System configuration, software updates, network configuration, firewall rules modification, hardware replacement, etc.

The bugreport should contain the answers for the previous questions and in addition at least the following information:

- Knot DNS version and type of installation (distribution package, from source, etc.)
- Operating system, platform, kernel version
- Relevant basic hardware information (processor, amount of memory, available network devices, etc.)
- Description of the bug
- Log output with the highest verbosity (category `any`, severity `debug`)
- Steps to reproduce the bug (if known)
- Backtrace (if the bug caused a crash or a hang; see the next section)

If possible, please provide a minimal configuration file and zone files which can be used to reproduce the bug.

6.2 Generating backtrace

Backtrace carries basic information about the state of the program and how the program got where it is. It helps determining the location of the bug in the source code.

If you run Knot DNS from distribution packages, make sure the debugging symbols for the package are installed. The symbols are usually distributed in a separate package.

There are several ways to get the backtrace. One possible way is to extract the backtrace from a core dump file. Core dump is a memory snapshot generated by the operating system when a process crashes. The generating of core dumps must be usually enabled:

```
$ ulimit -c unlimited                    # Enable unlimited core dump size
$ knotd ...                              # Reproduce the crash
...
$ gdb knotd <core-dump-file>             # Start gdb on the core dump
(gdb) info threads                       # Get a summary of all threads
(gdb) thread apply all bt full           # Extract backtrace from all threads
(gdb) quit
```

To save the backtrace into a file, the following GDB commands can be used:

```
(gdb) set pagination off
(gdb) set logging file backtrace.txt
(gdb) set logging on
(gdb) info threads
(gdb) thread apply all bt full
(gdb) set logging off
```

To generate a core dump of a running process, the *gcore* utility can be used:

```
$ gcore -o <output-file> $(pidof knotd)
```

Please note that core dumps can be intercepted by an error-collecting system service (systemd-coredump, ABRT, Apport, etc.). If you are using such a service, consult its documentation about core dump retrieval.

If the error is reproducible, it is also possible to start and inspect the server directly in the debugger:

```
$ gdb --args knotd -c /etc/knot.conf
(gdb) run
...
```

Alternatively, the debugger can be attached to a running server process. This is generally useful when troubleshooting a stuck process:

```
$ knotd ...
$ gdb --pid $(pidof knotd)
(gdb) continue
...
```

If you fail to get a backtrace of a running process using the previous method, you may try the single-purpose `pstack` utility:

```
$ pstack $(pidof knotd) > backtrace.txt
```

CHAPTER

SEVEN

CONFIGURATION REFERENCE

7.1 Description

Configuration files for Knot DNS use simplified YAML format. Simplified means that not all of the features are supported.

For the description of configuration items, we have to declare a meaning of the following symbols:

- *INT* – Integer
- *STR* – Textual string
- *HEXSTR* – Hexadecimal string (with `0x` prefix)
- *BOOL* – Boolean value (`on`/`off` or `true`/`false`)
- *TIME* – Number of seconds, an integer with possible time multiplier suffix (`s` ~ 1, `m` ~ 60, `h` ~ 3600 or `d` ~ 24 * 3600)
- *SIZE* – Number of bytes, an integer with possible size multiplier suffix (`B` ~ 1, `K` ~ 1024, `M` ~ 1024^2 or `G` ~ 1024^3)
- *BASE64* – Base64 encoded string
- *ADDR* – IPv4 or IPv6 address
- *DNAME* – Domain name
- ... – Multi-valued item, order of the values is preserved
- [ ] – Optional value
- | – Choice

There are 12 main sections (`module`, `server`, `control`, `log`, `statistics`, `keystore`, `policy`, `key`, `acl`, `remote`, `template`, and `zone`) and module sections with the `mod-` prefix. Most of the sections (excluding `server`, `control`, and `statistics`) are sequences of settings blocks. Each settings block begins with a unique identifier, which can be used as a reference from other sections (such identifier must be defined in advance).

A multi-valued item can be specified either as a YAML sequence:

```
address: [10.0.0.1, 10.0.0.2]
```

or as more single-valued items each on an extra line:

```
address: 10.0.0.1
address: 10.0.0.2
```

If an item value contains spaces or other special characters, it is necessary to enclose such value within double quotes `" "`.

7.2 Comments

A comment begins with a # character and is ignored during processing. Also each configuration section or sequence block allows a permanent comment using the `comment` item which is stored in the server beside the configuration.

7.3 Includes

Another configuration file or files, matching a pattern, can be included at the top level in the current file. If the path is not absolute, then it is considered to be relative to the current file. The pattern can be an arbitrary string meeting POSIX *glob* requirements, e.g. dir/*.conf. Matching files are processed in sorted order.

```
include: STR
```

7.4 Module section

Dynamic modules loading configuration.

Note: If configured with non-empty `--with-moduledir=path` parameter, all shared modules in this directory will be automatically loaded.

```
module:
  - id: STR
    file: STR
```

7.4.1 id

A module identifier in the form of the `mod-` prefix and module name suffix.

7.4.2 file

A path to a shared library file with the module implementation.

Default: `${libdir}/knot/modules-${version}`/module_name.so (or `${path}`/module_name.so if configured with `--with-moduledir=path`)

Warning: If the path is not absolute, the library is searched in the set of system directories. See `man dlopen` for more details.

7.5 Server section

General options related to the server.

```
server:
    identity: [STR]
    version: [STR]
    nsid: [STR|HEXSTR]
    rundir: STR
```

```
user: STR[:STR]
pidfile: STR
udp-workers: INT
tcp-workers: INT
background-workers: INT
async-start: BOOL
tcp-handshake-timeout: TIME
tcp-idle-timeout: TIME
tcp-reply-timeout: TIME
max-tcp-clients: INT
max-udp-payload: SIZE
max-ipv4-udp-payload: SIZE
max-ipv6-udp-payload: SIZE
listen: ADDR[@INT] ...
```

7.5.1 identity

An identity of the server returned in the response to the query for TXT record `id.server.` or `hostname.bind.` in the CHAOS class (see RFC 4892). Set empty value to disable.

Default: FQDN hostname

7.5.2 version

A version of the server software returned in the response to the query for TXT record `version.server.` or `version.bind.` in the CHAOS class (see RFC 4892). Set empty value to disable.

Default: server version

7.5.3 nsid

A DNS name server identifier (see RFC 5001). Set empty value to disable.

Default: FQDN hostname

7.5.4 rundir

A path for storing run-time data (PID file, unix sockets, etc.).

Default: `${localstatedir}/run/knot` (configured with `--with-rundir=path`)

7.5.5 user

A system user with an optional system group (`user:group`) under which the server is run after starting and binding to interfaces. Linux capabilities are employed if supported.

Default: root:root

7.5.6 pidfile

A PID file location.

Default: *rundir*/knot.pid

7.5.7 udp-workers

A number of UDP workers (threads) used to process incoming queries over UDP.

Default: auto-estimated optimal value based on the number of online CPUs

7.5.8 tcp-workers

A number of TCP workers (threads) used to process incoming queries over TCP.

Default: auto-estimated optimal value based on the number of online CPUs

7.5.9 background-workers

A number of workers (threads) used to execute background operations (zone loading, zone updates, etc.).

Default: auto-estimated optimal value based on the number of online CPUs

7.5.10 async-start

If enabled, server doesn't wait for the zones to be loaded and starts responding immediately with SERVFAIL answers until the zone loads.

Default: off

7.5.11 tcp-handshake-timeout

Maximum time between newly accepted TCP connection and the first query. This is useful to disconnect inactive connections faster than connections that already made at least 1 meaningful query.

Default: 5

7.5.12 tcp-idle-timeout

Maximum idle time between requests on a TCP connection. This also limits receiving of a single query, each query must be received in this time limit.

Default: 20

7.5.13 tcp-reply-timeout

Maximum time to wait for an outgoing connection or for a reply to an issued request (SOA, NOTIFY, AXFR...).

Default: 10

7.5.14 max-tcp-clients

A maximum number of TCP clients connected in parallel, set this below the file descriptor limit to avoid resource exhaustion.

Default: 100

7.5.15 max-udp-payload

Maximum EDNS0 UDP payload size default for both IPv4 and IPv6.

Default: 4096

7.5.16 max-ipv4-udp-payload

Maximum EDNS0 UDP payload size for IPv4.

Default: 4096

7.5.17 max-ipv6-udp-payload

Maximum EDNS0 UDP payload size for IPv6.

Default: 4096

7.5.18 listen

One or more IP addresses where the server listens for incoming queries. Optional port specification (default is 53) can be appended to each address using `@` separator. Use `0.0.0.0` for all configured IPv4 addresses or `::` for all configured IPv6 addresses.

Default: not set

7.6 Key section

Shared TSIG keys used to authenticate communication with the server.

```
key:
  - id: DNAME
    algorithm: hmac-md5 | hmac-sha1 | hmac-sha224 | hmac-sha256 | hmac-sha384 |␣
↪hmac-sha512
    secret: BASE64
```

7.6.1 id

A key name identifier.

Note: This value MUST be exactly the same as the name of the TSIG key on the opposite master/slave server(s).

7.6.2 algorithm

A key algorithm.

Default: not set

7.6.3 secret

Shared key secret.

Default: not set

7.7 ACL section

Access control list rule definitions. The ACLs are used to match incoming connections to allow or deny requested operation (zone transfer request, DDNS update, etc.).

```
acl:
  - id: STR
    address: ADDR[/INT] | ADDR-ADDR ...
    key: key_id ...
    action: notify | transfer | update ...
    deny: BOOL
```

7.7.1 id

An ACL rule identifier.

7.7.2 address

An ordered list of IP addresses, network subnets, or network ranges. The query must match one of them. Empty value means that address match is not required.

Default: not set

7.7.3 key

An ordered list of *reference*s to TSIG keys. The query must match one of them. Empty value means that TSIG key is not required.

Default: not set

7.7.4 action

An ordered list of allowed (or denied) actions.

Possible values:

- `transfer` – Allow zone transfer
- `notify` – Allow incoming notify
- `update` – Allow zone updates

Default: not set

7.7.5 deny

If enabled, instead of allowing, deny the specified *action*, *address*, *key*, or combination if these items. If no action is specified, deny all actions.

Default: off

7.8 Control section

Configuration of the server control interface.

```
control:
    listen: STR
    timeout: TIME
```

7.8.1 listen

A UNIX socket path where the server listens for control commands.

Default: *rundir*/knot.sock

7.8.2 timeout

Maximum time the control socket operations can take. Set 0 for infinity.

Default: 5

7.9 Statistics section

Periodic server statistics dumping.

```
statistics:
    timer: TIME
    file: STR
    append: BOOL
```

7.9.1 timer

A period after which all available statistics metrics will by written to the *file*.

Default: not set

7.9.2 file

A file path of statistics output in the YAML format.

Default: *rundir*/stats.yaml

7.9.3 append

If enabled, the output will be appended to the *file* instead of file replacement.

Default: off

7.10 Keystore section

DNSSEC keystore configuration.

```
keystore:
  - id: STR
    backend: pem | pkcs11
    config: STR
```

7.10.1 id

A keystore identifier.

7.10.2 backend

A key storage backend type. A directory with PEM files or a PKCS #11 storage.

Default: pem

7.10.3 config

A backend specific configuration. A directory with PEM files (the path can be specified as a relative path to *kasp-db*) or a configuration string for PKCS #11 storage.

Note: Example configuration string for PKCS #11:

```
"pkcs11:token=knot;pin-value=1234 /usr/lib64/pkcs11/libsofthsm2.so"
```

Default: *kasp-db*/keys

7.11 Submission section

Parameters of KSK submission checks.

```
submission:
  - id: STR
    parent: remote_id ...
    check-interval: TIME
    timeout: TIME
```

7.11.1 id

A submission identifier.

7.11.2 parent

A list of *references* to parent's DNS servers to be checked for presence of corresponding DS records in case of KSK submission. All of them must have corresponding DS for the rollover to continue. If none specified, the rollover must be pushed forward manually.

Default: not set

7.11.3 check-interval

Interval for periodic checks of DS resence on parent's DNS servers, in case of KSK submission.

Default: 1 hour

7.11.4 timeout

After this period, the KSK submission is automatically considered successful, even if all the checks were negative or no parents are configured. Set 0 for infinity.

Default: infinity

7.12 Policy section

DNSSEC policy configuration.

```
policy:
  - id: STR
    keystore: STR
    manual: BOOL
    single-type-signing: BOOL
    algorithm: dsa | rsasha1 | dsa-nsec3-sha1 | rsasha1-nsec3-sha1 | rsasha256 |␣
↪rsasha512 | ecdsap256sha256 | ecdsap384sha384
    ksk-size: SIZE
    zsk-size: SIZE
    ksk-shared: BOOL
    dnskey-ttl: TIME
    zsk-lifetime: TIME
    ksk-lifetime: TIME
    propagation-delay: TIME
    rrsig-lifetime: TIME
    rrsig-refresh: TIME
    nsec3: BOOL
    nsec3-iterations: INT
    nsec3-salt-length: INT
    nsec3-salt-lifetime: TIME
    ksk-submission: submission_id
```

7.12.1 id

A policy identifier.

7.12.2 keystore

A *reference* to a keystore holding private key material for zones. A special *default* value can be used for the default keystore settings.

Default: default

7.12.3 manual

If enabled, automatic key management is not used.

Default: off

7.12.4 single-type-signing

If enabled, Single-Type Signing Scheme is used in the automatic key management mode.

Note: Because key rollover is not supported yet, just one combined signing key is generated if none is available.

Default: off

7.12.5 algorithm

An algorithm of signing keys and issued signatures.

Default: ecdsap256sha256

7.12.6 ksk-size

A length of newly generated KSK (Key Signing Key) or CSK (Combined Signing Key) keys.

Default: 1024 (dsa*), 2048 (rsa*), 256 (ecdsap256*), 384 (ecdsap384*)

7.12.7 zsk-size

A length of newly generated ZSK (Zone Signing Key) keys.

Default: see default for *ksk-size*

7.12.8 ksk-shared

If enabled, all zones with this policy assigned will share one KSK.

Default: off

7.12.9 dnskey-ttl

A TTL value for DNSKEY records added into zone apex.

Default: zone SOA TTL

Note: has infuence over ZSK key lifetime

7.12.10 zsk-lifetime

A period between ZSK publication and the next rollover initiation.

Default: 30 days

Note: ZSK key lifetime is also infuenced by propagation-delay and dnskey-ttl

---

7.12.11 ksk-lifetime

A period between KSK publication and the next rollover initiation.

Default: infinity

Note: KSK key lifetime is also infuenced by propagation-delay, dnskey-ttl, and KSK submission delay.

The default infinite value causes no KSK rollover as a result.

This applies for CSK lifetime if single-type-signing is enabled.

7.12.12 propagation-delay

An extra delay added for each key rollover step. This value should be high enough to cover propagation of data from the master server to all slaves.

Default: 1 day

Note: has infuence over ZSK key lifetime

7.12.13 rrsig-lifetime

A validity period of newly issued signatures.

Default: 14 days

7.12.14 rrsig-refresh

A period how long before a signature expiration the signature will be refreshed.

Default: 7 days

7.12.15 nsec3

Specifies if NSEC3 will be used instead of NSEC.

Default: off

7.12.16 nsec3-iterations

A number of additional times the hashing is performed.

Default: 5

7.12.17 nsec3-salt-length

A length of a salt field in octets, which is appended to the original owner name before hashing.

Default: 8

7.12.18 nsec3-salt-lifetime

A validity period of newly issued salt field.

Default: 30 days

7.12.19 ksk-submission

A reference to *submission* section holding parameters of KSK submittion checks.

Default: not set

7.13 Remote section

Definitions of remote servers for outgoing connections (source of a zone transfer, target for a notification, etc.).

```
remote:
  - id: STR
    address: ADDR[@INT] ...
    via: ADDR[@INT] ...
    key: key_id
```

7.13.1 id

A remote identifier.

7.13.2 address

An ordered list of destination IP addresses which are used for communication with the remote server. The addresses are tried in sequence unless the operation is successful. Optional destination port (default is 53) can be appended to the address using `@` separator.

Default: not set

7.13.3 via

An ordered list of source IP addresses. The first address with the same family as the destination address is used. Optional source port (default is random) can be appended to the address using `@` separator.

Default: not set

7.13.4 key

A *reference* to the TSIG key which is used to authenticate the communication with the remote server.

Default: not set

7.14 Template section

A template is a shareable zone setting which can be used for configuration of many zones in one place. A special default template (with the *default* identifier) can be used for global querying configuration or as an implicit configuration if a zone doesn't have another template specified.

```
template:
  - id: STR
    timer-db: STR
    max-timer-db-size: SIZE
    journal-db: STR
    journal-db-mode: robust | asynchronous
    max-journal-db-size: SIZE
    kasp-db: STR
    max-kasp-db-size: SIZE
    global-module: STR/STR ...
    # All zone options (excluding 'template' item)
```

7.14.1 id

A template identifier.

7.14.2 timer-db

Specifies a path of the persistent timer database. The path can be specified as a relative path to the *default* template *storage*.

Note: This option is only available in the *default* template.

Default: *storage*/timers

7.14.3 max-timer-db-size

Hard limit for the timer database maximum size.

Note: This option is only available in the *default* template.

Default: 100 MiB

7.14.4 journal-db

Specifies a path of the persistent journal database. The path can be specified as a relative path to the *default* template *storage*.

Note: This option is only available in the *default* template.

Default: *storage*/journal

7.14.5 journal-db-mode

Specifies journal LMDB backend configuration, which influences performance and durability.

Possible values:

- `robust` – The journal DB disk sychronization ensures DB durability but is generally slower
- `asynchronous` – The journal DB disk synchronization is optimized for better perfomance at the expense of lower DB durability; this mode is recommended only on slave nodes with many zones

Note: This option is only available in the *default* template.

Default: robust

7.14.6 max-journal-db-size

Hard limit for the common journal DB. There is no cleanup logic in journal to recover from reaching this limit: journal simply starts refusing changes across all zones. Decreasing this value has no effect if lower than actual DB file size.

It is recommended to limit *max-journal-usage* per-zone instead of max-journal-size in most cases. Please keep this value larger than the sum of all zones' journal usage limits.

This value also influences server's usage of virtual memory.

Note: This option is only available in the *default* template.

Default: 20 GiB (1 GiB for 32-bit)

7.14.7 kasp-db

A KASP database path. Non-absolute path is relative to *storage*.

Default: *storage*/keys

Note: This option is only available in the *default* template.

7.14.8 max-kasp-db-size

Hard limit for the KASP database maximum size.

Note: This option is only available in the *default* template.

Default: 500 MiB

7.14.9 global-module

An ordered list of references to query modules in the form of *module_name* or *module_name/module_id*. These modules apply to all queries.

Note: This option is only available in the *default* template.

Default: not set

7.15 Zone section

Definition of zones served by the server.

```
zone:
  - domain: DNAME
    template: template_id
    storage: STR
    file: STR
    master: remote_id ...
    ddns-master: remote_id
    notify: remote_id ...
    acl: acl_id ...
    semantic-checks: BOOL
    disable-any: BOOL
    zonefile-sync: TIME
    ixfr-from-differences: BOOL
```

```
max-journal-usage: SIZE
max-journal-depth: INT
max-zone-size : SIZE
dnssec-signing: BOOL
dnssec-policy: STR
request-edns-option: INT:[HEXSTR]
serial-policy: increment | unixtime
min-refresh-interval: TIME
max-refresh-interval: TIME
module: STR/STR ...
```

7.15.1 domain

A zone name identifier.

7.15.2 template

A *reference* to a configuration template.

Default: not set or *default* (if the template exists)

7.15.3 storage

A data directory for storing zone files, journal database, and timers database.

Default: `${localstatedir}/lib/knot` (configured with `--with-storage=path`)

7.15.4 file

A path to the zone file. Non-absolute path is relative to *storage*. It is also possible to use the following formatters:

- `%c[N]` or `%c[N-M]` – means the *N*th character or a sequence of characters beginning from the *N*th and ending with the *M*th character of the textual zone name (see `%s`). The indexes are counted from 0 from the left. All dots (including the terminal one) are considered. If the character is not available, the formatter has no effect.
- `%l[N]` – means the *N*th label of the textual zone name (see `%s`). The index is counted from 0 from the right (0 ~ TLD). If the label is not available, the formatter has no effect.
- `%s` – means the current zone name in the textual representation. The zone name doesn't include the terminating dot (the result for the root zone is the empty string!).
- `%%` – means the `%` character

Warning: Beware of special characters which are escaped or encoded in the \DDD form where DDD is corresponding decimal ASCII code.

Default: *storage*/`%s`.zone

7.15.5 master

An ordered list of *references* to zone master servers.

Default: not set

7.15.6 ddns-master

A *reference* to zone primary master server. If not specified, the first *master* server is used.

Default: not set

7.15.7 notify

An ordered list of *references* to remotes to which notify message is sent if the zone changes.

Default: not set

7.15.8 acl

An ordered list of *references* to ACL rules which can allow or disallow zone transfers, updates or incoming notifies.

Default: not set

7.15.9 semantic-checks

If enabled, extra zone file semantic checks are turned on.

Several checks are enabled by default and cannot be turned off. An error in mandatory checks causes zone not to be loaded. An error in extra checks is logged only.

Mandatory checks:

- An extra record together with CNAME record (except for RRSIG and DS)
- SOA record missing in the zone (RFC 1034)
- DNAME records having records under it (DNAME children) (RFC 2672)

Extra checks:

- Missing NS record at the zone apex
- Missing glue A or AAAA records
- Broken or non-cyclic NSEC(3) chain
- Wrong NSEC(3) type bitmap
- Multiple NSEC records at the same node
- Missing NSEC records at authoritative nodes
- NSEC3 insecure delegation that is not part of Opt-out span
- Wrong original TTL value in NSEC3 records
- Wrong RDATA TTL value in RRSIG record
- Signer name in RRSIG RR not the same as in DNSKEY
- Signed RRSIG
- Wrong key flags or wrong key in RRSIG record (not the same as ZSK)

Default: off

7.15.10 disable-any

If enabled, all authoritative ANY queries sent over UDP will be answered with an empty response and with the TC bit set. Use this option to minimize the risk of DNS reflection attack.

Default: off

7.15.11 zonefile-sync

The time after which the current zone in memory will be synced with a zone file on the disk (see *file*). The server will serve the latest zone even after a restart using zone journal, but the zone file on the disk will only be synced after `zonefile-sync` time has expired (or after manual zone flush). This is applicable when the zone is updated via IXFR, DDNS or automatic DNSSEC signing. In order to completely disable automatic zonefile synchronization, set the value to -1. In that case, it is still possible to force a manual zone flush using the `-f` option.

Note: If you are serving large zones with frequent updates where the immediate sync with a zone file is not desirable, increase the value.

Warning: If the zone file is not up-to-date, the zone should be flushed before its zone file editation or the SOA record must be untouched after editation. Otherwise the journal can't be applied.

Default: 0 (immediate)

7.15.12 ixfr-from-differences

If enabled, the server creates zone differences from changes you made to the zone file upon server reload. This option is relevant only if the server is a master server for the zone.

Note: This option has no effect with enabled *dnssec-signing*.

Default: off

7.15.13 max-journal-usage

Policy how much space in journal DB will the zone's journal occupy.

Default: 100 MiB

Note: Journal DB may grow far above the sum of max-journal-usage across all zones, because of DB free space fragmentation.

7.15.14 max-journal-depth

Maximum history length of journal.

Minimum: 2

Default: 2^64

7.15.15 max-zone-size

Maximum size of the zone. The size is measured as size of the zone records in wire format without compression. The limit is enforced for incoming zone transfers and dynamic updates.

For incremental transfers (IXFR), the effective limit for the total size of the records in the transfer is twice the configured value. However the final size of the zone must satisfy the configured value.

Default: 2^64

7.15.16 dnssec-signing

If enabled, automatic DNSSEC signing for the zone is turned on.

Note: Cannot be enabled on a slave zone.

Default: off

7.15.17 dnssec-policy

A *reference* to DNSSEC signing policy. A special *default* value can be used for the default policy settings.

Required

7.15.18 request-edns-option

An arbitrary EDNS0 option which is included into a server request (AXFR, IXFR, SOA, or NOTIFY). The value is in the option_code:option_data format.

Default: not set

7.15.19 serial-policy

Specifies how the zone serial is updated after a dynamic update or automatic DNSSEC signing. If the serial is changed by the dynamic update, no change is made.

Possible values:

- `increment` – The serial is incremented according to serial number arithmetic
- `unixtime` – The serial is set to the current unix time

Note: If your serial was in other than unix time format, be careful with the transition to unix time. It may happen that the new serial will be 'lower' than the old one. If this is the case, the transition should be done by hand (see RFC 1982).

Default: increment

7.15.20 min-refresh-interval

Forced minimum zone refresh interval to avoid flooding master.

Default: 2

7.15.21 max-refresh-interval

Forced maximum zone refresh interval.

Default: not set

7.15.22 module

An ordered list of references to query modules in the form of *module_name* or *module_name/module_id*. These modules apply only to the current zone queries.

Default: not set

7.16 Logging section

Server can be configured to log to the standard output, standard error output, syslog (or systemd journal if systemd is enabled) or into an arbitrary file.

There are 6 logging severity levels:

- `critical` – Non-recoverable error resulting in server shutdown
- `error` – Recoverable error, action should be taken
- `warning` – Warning that might require user action
- `notice` – Server notice or hint
- `info` – Informational message
- `debug` – Debug messages (must be turned on at compile time)

In the case of missing log section, `warning` or more serious messages will be logged to both standard error output and syslog. The `info` and `notice` messages will be logged to standard output.

```
log:
  - target: stdout | stderr | syslog | STR
    server: critical | error | warning | notice | info | debug
    control: critical | error | warning | notice | info | debug
    zone: critical | error | warning | notice | info | debug
    any: critical | error | warning | notice | info | debug
```

7.16.1 target

A logging output.

Possible values:

- `stdout` – Standard output
- `stderr` – Standard error output
- `syslog` – Syslog
- *file_name* – File

7.16.2 server

Minimum severity level for messages related to general operation of the server that are logged.

Default: not set

7.16.3 control

Minimum severity level for messages related to server control that are logged.

Default: not set

7.16.4 zone

Minimum severity level for messages related to zones that are logged.

Default: not set

7.16.5 any

Minimum severity level for all message types that are logged.

Default: not set

CHAPTER

EIGHT

MODULES

8.1 dnsproxy – Tiny DNS proxy

The module forwards all queries, or all specific zone queries if configured per zone, to the indicated server for resolution. If configured in the fallback mode, only localy unsatisfied queries are forwarded. I.e. a tiny DNS proxy. There are several uses of this feature:

- A substitute public-facing server in front of the real one
- Local zones (poor man's "views"), rest is forwarded to the public-facing server
- etc.

Note: The module does not alter the query/response as the resolver would, and the original transport protocol is kept as well.

8.1.1 Example

The configuration is straightforward and just a single remote server is required:

```
remote:
  - id: hidden
    address: 10.0.1.1

mod-dnsproxy:
  - id: default
    remote: hidden
    fallback: on

template:
  - id: default
    global-module: mod-dnsproxy/default

zone:
  - domain: local.zone
```

When clients query for anything in the `local.zone`, they will be responded to locally. The rest of the requests will be forwarded to the specified server (`10.0.1.1` in this case).

8.1.2 Module reference

```
mod-dnsproxy:
  - id: STR
    remote: remote_id
    timeout: INT
```

```
    fallback: BOOL
    catch-nxdomain: BOOL
```

id

A module identifier.

remote

A *reference* to a remote server where the queries are forwarded to.

Required

timeout

A remote response timeout in milliseconds.

Default: 500

fallback

If enabled, localy unsatisfied queries leading to REFUSED (no zone) are forwarded. If disabled, all queries are directly forwarded without any local attempts to resolve them.

Default: on

catch-nxdomain

If enabled, localy unsatisfied queries leading to NXDOMAIN are forwarded. This option is only relevant in the fallback mode.

Default: off

8.2 `dnstap` – Dnstap traffic logging

A module for query and response logging based on the dnstap library. You can capture either all or zone-specific queries and responses; usually you want to do the former.

8.2.1 Example

The configuration comprises only a *sink* path parameter, which can be either a file or a UNIX socket:

```
mod-dnstap:
  - id: capture_all
    sink: /tmp/capture.tap

template:
  - id: default
    global-module: mod-dnstap/capture_all
```

Note: To be able to use a Unix socket you need an external program to create it. Knot DNS connects to it as a client using the libfstrm library. It operates exactly like syslog. See here for more details.

Note: Dnstap log files can also be created or read using `kdig`.

8.2.2 Module reference

For all queries logging, use this module in the *default* template. For zone-specific logging, use this module in the proper zone configuration.

```
mod-dnstap:
  - id: STR
    sink: STR
    identity: STR
    version: STR
    log-queries: BOOL
    log-responses: BOOL
```

id

A module identifier.

sink

A sink path, which can be either a file or a UNIX socket when prefixed with `unix:`.

Required

Warning: File is overwritten on server startup or reload.

identity

A DNS server identity. Set empty value to disable.

Default: FQDN hostname

version

A DNS server version. Set empty value to disable.

Default: server version

log-queries

If enabled, query messages will be logged.

Default: on

log-responses

If enabled, response messages will be logged.

Default: on

8.3 noudp — No UDP response

The module sends empty truncated response to any UDP query. TCP queries are not affected.

8.3.1 Example

To enable this module globally, you need to add something like the following to the configuration file:

```
template:
  - id: default
    global-module: mod-noudp
```

Note: This module is not configurable.

8.4 onlinesign — Online DNSSEC signing

The module provides online DNSSEC signing. Instead of pre-computing the zone signatures when the zone is loaded into the server or instead of loading an externally signed zone, the signatures are computed on-the-fly during answering.

The main purpose of the module is to enable authenticated responses with zones which use other dynamic module (e.g., automatic reverse record synthesis) because these zones cannot be pre-signed. However, it can be also used as a simple signing solution for zones with low traffic and also as a protection against zone content enumeration (zone walking).

In order to minimize the number of computed signatures per query, the module produces a bit different responses from the responses that would be sent if the zone was pre-signed. Still, the responses should be perfectly valid for a DNSSEC validating resolver.

Differences from statically signed zones:

- The NSEC records are constructed as Minimally Covering NSEC Records (see Appendix A in **RFC 7129**). Therefore the generated domain names cover the complete domain name space in the zone's authority.
- NXDOMAIN responses are promoted to NODATA responses. The module proves that the query type does not exist rather than that the domain name does not exist.
- Domain names matching a wildcard are expanded. The module pretends and proves that the domain name exists rather than proving a presence of the wildcard.

Records synthesized by the module:

- DNSKEY record is synthesized in the zone apex and includes public key material for the active signing key.
- NSEC records are synthesized as needed.
- RRSIG records are synthesized for authoritative content of the zone.

Known issues:

- The delegations are not signed correctly.
- Some CNAME records are not signed correctly.

- The automatic policy-based key rotation does not work. The rotation events are invoked just at server (re)load.

Limitations:

- Online-sign module always enforces Single-Type Signing scheme.
- Only one active signing key can be used.
- Key rollover is not possible.
- The NSEC records may differ for one domain name if queried for different types. This is an implementation shortcoming as the dynamic modules cooperate loosely. Possible synthesis of a type by other module cannot be predicted. This dissimilarity should not affect response validation, even with validators performing aggressive negative caching.
- The NSEC proofs will work well with other dynamic modules only if the modules synthesize only A and AAAA records. If synthesis of other type is required, please, report this information to Knot DNS developers.

8.4.1 Example

- Enable the module in the zone configuration with the default signing policy:

```
zone:
  - domain: example.com
    module: mod-onlinesign
```

  Or with an explicit signing policy:

```
policy:
  - id: rsa
    algorithm: RSASHA256
    zsk-size: 2048

mod-onlinesign:
  - id: explicit
    policy: rsa

zone:
  - domain: example.com
    module: mod-onlinesign/explicit
```

  Or use manual policy in an analogous manner, see *Manual key management*.

  Note: Only id, manual, keystore, algorithm, ksk-size, and rrsig-lifetime policy items are relevant to this module. If no rrsig-lifetime is configured, the default value is 25 hours.

- Make sure the zone is not signed and also that the automatic signing is disabled. All is set, you are good to go. Reload (or start) the server:

```
$ knotc reload
```

The following example stacks the online signing with reverse record synthesis module:

```
mod-synthrecord:
  - id: lan-forward
    type: forward
    prefix: ip-
    ttl: 1200
    network: 192.168.100.0/24
```

```
zone:
  - domain: corp.example.net
    module: [mod-synthrecord/lan-forward, mod-onlinesign]
```

8.4.2 Module reference

```
mod-onlinesign:
  - id: STR
    policy: STR
```

id

A module identifier.

policy

A *reference* to DNSSEC signing policy. A special *default* value can be used for the default policy settings.

8.5 `rosedb` – Static resource records

The module provides a mean to override responses for certain queries before the record is searched in the available zones. The module comes with the `rosedb_tool` tool used to manipulate the database of static records.

For example, let's suppose we have a database of following records:

```
myrecord.com.      3600 IN A 127.0.0.1
www.myrecord.com.  3600 IN A 127.0.0.2
ipv6.myrecord.com. 3600 IN AAAA ::1
```

And we query the nameserver with the following:

```
$ kdig IN A myrecord.com
  ... returns NOERROR, 127.0.0.1
$ kdig IN A www.myrecord.com
  ... returns NOERROR, 127.0.0.2
$ kdig IN A stuff.myrecord.com
  ... returns NOERROR, 127.0.0.1
$ kdig IN AAAA myrecord.com
  ... returns NOERROR, NODATA
$ kdig IN AAAA ipv6.myrecord.com
  ... returns NOERROR, ::1
```

An entry in the database matches anything at the same or a lower domain level, i.e. 'myrecord.com' matches 'a.a.myrecord.com' as well. This can be utilized to create catch-all entries.

You can also add authority information for the entries, provided you create SOA + NS records for a name, like so:

```
myrecord.com.      3600 IN SOA master host 1 3600 60 3600 3600
myrecord.com.      3600 IN NS ns1.myrecord.com.
myrecord.com.      3600 IN NS ns2.myrecord.com.
ns1.myrecord.com. 3600 IN A 127.0.0.1
ns2.myrecord.com. 3600 IN A 127.0.0.2
```

In this case, the responses will:

1. Be authoritative (AA flag set)

2. Provide an authority section (SOA + NS)
3. Be NXDOMAIN if the name is found *(i.e. the 'IN AAAA myrecord.com' from the example)*, but not the RR type *(this is to allow the synthesis of negative responses)*

The SOA record applies only to the 'myrecord.com.', not to any other record (not even those of its subdomains). From this point of view, all records in the database are unrelated and not hierarchical. The idea is to provide subtree isolation for each entry.

In addition, the module is able to log matching queries via remote syslog if you specify a syslog address endpoint and an optional string code.

8.5.1 Example

- Create the entries in the database:

```
$ mkdir /tmp/static_rrdb
$ # No logging
$ rosedb_tool /tmp/static_rrdb add myrecord.com. A 3600 "127.0.0.1" "-" "-"
$ # Logging as 'www_query' to Syslog at 10.0.0.1
$ rosedb_tool /tmp/static_rrdb add www.myrecord.com. A 3600 "127.0.0.1" \
                                                  "www_query" "10.0.0.1"
$ # Logging as 'ipv6_query' to Syslog at 10.0.0.1
$ rosedb_tool /tmp/static_rrdb add ipv6.myrecord.com. AAAA 3600 "::1" \
                                              "ipv6_query" "10.0.0.1"
$ # Verify settings
$ rosedb_tool /tmp/static_rrdb list
www.myrecord.com.       A RDATA=10B     www_query       10.0.0.1
ipv6.myrecord.com.      AAAA RDATA=22B  ipv6_query      10.0.0.1
myrecord.com.           A RDATA=10B     -               -
```

Note: The database may be modified later on while the server is running.

- Configure the query module:

```
mod-rosedb:
  - id: default
    dbdir: /tmp/static_rrdb

template:
  - id: default
    global-module: mod-rosedb/default
```

The module accepts just one parameter – the path to the directory where the database will be stored.

- Start the server:

```
$ knotd -c knot.conf
```

- Verify the running instance:

```
$ kdig @127.0.0.1#6667 A myrecord.com
```

8.5.2 Module reference

```
mod-rosedb:
  - id: STR
    dbdir: STR
```

id

A module identifier.

dbdir

A path to the directory where the database is stored.

Required

8.6 `rrl` — Response rate limiting

Response rate limiting (RRL) is a method to combat DNS reflection amplification attacks. These attacks rely on the fact that source address of a UDP query can be forged, and without a worldwide deployment of BCP38, such a forgery cannot be prevented. An attacker can use a DNS server (or multiple servers) as an amplification source and can flood a victim with a large number of unsolicited DNS responses. The RRL lowers the amplification factor of these attacks by sending some of the responses as truncated or by dropping them altogether.

Note: The module introduces two statistics counters. The number of slipped and dropped responses.

8.6.1 Example

You can enable RRL by setting the module globally or per zone.

```
mod-rrl:
  - id: default
    rate-limit: 200   # Allow 200 resp/s for each flow
    slip: 2           # Every other response slips

template:
  - id: default
    global-module: mod-rrl/default   # Enable RRL globally
```

8.6.2 Module reference

```
mod-rrl:
  - id: STR
    rate-limit: INT
    slip: INT
    table-size: INT
    whitelist: ADDR[/INT] | ADDR-ADDR ...
```

id

A module identifier.

rate-limit

Rate limiting is based on the token bucket scheme. A rate basically represents a number of tokens available each second. Each response is processed and classified (based on several discriminators, e.g. source netblock, query type, zone name, rcode, etc.). Classified responses are then hashed and assigned to a bucket containing number

of available tokens, timestamp and metadata. When available tokens are exhausted, response is dropped or sent as truncated (see *slip*). Number of available tokens is recalculated each second.

Required

table-size

Size of the hash table in a number of buckets. The larger the hash table, the lesser the probability of a hash collision, but at the expense of additional memory costs. Each bucket is estimated roughly to 32 bytes. The size should be selected as a reasonably large prime due to better hash function distribution properties. Hash table is internally chained and works well up to a fill rate of 90 %, general rule of thumb is to select a prime near 1.2 * maximum_qps.

Default: 393241

slip

As attacks using DNS/UDP are usually based on a forged source address, an attacker could deny services to the victim's netblock if all responses would be completely blocked. The idea behind SLIP mechanism is to send each N^{th} response as truncated, thus allowing client to reconnect via TCP for at least some degree of service. It is worth noting, that some responses can't be truncated (e.g. SERVFAIL).

- Setting the value to **0** will cause that all rate-limited responses will be dropped. The outbound bandwidth and packet rate will be strictly capped by the *rate-limit* option. All legitimate requestors affected by the limit will face denial of service and will observe excessive timeouts. Therefore this setting is not recommended.
- Setting the value to **1** will cause that all rate-limited responses will be sent as truncated. The amplification factor of the attack will be reduced, but the outbound data bandwidth won't be lower than the incoming bandwidth. Also the outbound packet rate will be the same as without RRL.
- Setting the value to **2** will cause that half of the rate-limited responses will be dropped, the other half will be sent as truncated. With this configuration, both outbound bandwidth and packet rate will be lower than the inbound. On the other hand, the dropped responses enlarge the time window for possible cache poisoning attack on the resolver.
- Setting the value to anything **larger than 2** will keep on decreasing the outgoing rate-limited bandwidth, packet rate, and chances to notify legitimate requestors to reconnect using TCP. These attributes are inversely proportional to the configured value. Setting the value high is not advisable.

Default: 1

whitelist

A list of IP addresses, network subnets, or network ranges to exempt from rate limiting. Empty list means that no incoming connection will be white-listed.

Default: not set

8.7 `stats` — Query statistics

The module extends server statistics with incoming DNS request and corresponding response counters, such as used network protocol, total number of responded bytes, etc (see module reference for full list of supported counters). This module should be configured as the last module.

Note: Server initiated communication (outgoing NOTIFY, incoming *XFR,...) is not counted by this module.

Note: Leading 16-bit message size over TCP is not considered.

8.7.1 Example

Common statistics with default module configuration:

```
template:
  - id: default
    global-module: mod-stats
```

Per zone statistics with explicit module configuration:

```
mod-stats:
  - id: custom
    edns-presence: on
    query-type: on

template:
  - id: default
    module: mod-stats/custom
```

8.7.2 Module reference

```
mod-stats:
  - id: STR
    request-protocol: BOOL
    server-operation: BOOL
    request-bytes: BOOL
    response-bytes: BOOL
    edns-presence: BOOL
    flag-presence: BOOL
    response-code: BOOL
    reply-nodata: BOOL
    query-type: BOOL
    query-size: BOOL
    reply-size: BOOL
```

id

A module identifier.

request-protocol

If enabled, all incoming requests are counted by the network protocol:

- udp4 - UDP over IPv4
- tcp4 - TCP over IPv4
- udp6 - UDP over IPv6
- tcp6 - TCP over IPv6

Default: on

server-operation

If enabled, all incoming requests are counted by the server operation. The server operation is based on message header OpCode and message query (meta) type:

- query - Normal query operation
- update - Dynamic update operation
- notify - NOTIFY request operation
- axfr - Full zone transfer operation
- ixfr - Incremental zone transfer operation
- invalid - Invalid server operation

Default: on

request-bytes

If enabled, all incoming request bytes are counted by the server operation:

- query - Normal query bytes
- update - Dynamic update bytes
- other - Other request bytes

Default: on

response-bytes

If enabled, outgoing response bytes are counted by the server operation:

- reply - Normal response bytes
- transfer - Zone transfer bytes
- other - Other response bytes

> **Warning:** Dynamic update response bytes are not counted by this module.

Default: on

edns-presence

If enabled, EDNS pseudo section presence is counted by the message direction:

- request - EDNS present in request
- response - EDNS present in response

Default: off

flag-presence

If enabled, some message header flags are counted:

- TC - Truncated Answer in response
- DO - DNSSEC OK in request

Default: off

response-code

If enabled, outgoing response code is counted:

- NOERROR
- ...
- NOTZONE
- BADVERS
- ...
- BADCOOKIE
- other - All other codes

Note: In the case of multi-message zone transfer response, just one counter is incremented.

Warning: Dynamic update response code is not counted by this module.

Default: on

reply-nodata

If enabled, NODATA pseudo RCODE (see RFC 2308, Section 2.2) is counted by the query type:

- A
- AAAA
- other - All other types

Default: off

query-type

If enabled, normal query type is counted:

- A (TYPE1)
- ...
- TYPE65
- SPF (TYPE99)
- ...
- TYPE110
- ANY (TYPE255)
- ...
- TYPE260
- other - All other types

Note: Not all assigned meta types (IXFR, AXFR,...) have their own counters, because such types are not processed as normal query.

Default: off

query-size

If enabled, normal query message size distribution is counted by the size range in bytes:

- 0-15
- 16-31
- ...
- 272-287
- 288-65535

Default: off

reply-size

If enabled, normal reply message size distribution is counted by the size range in bytes:

- 0-15
- 16-31
- ...
- 4080-4095
- 4096-65535

Default: off

8.8 `synthrecord` – Automatic forward/reverse records

This module is able to synthesize either forward or reverse records for a given prefix and subnet.

Records are synthesized only if the query can't be satisfied from the zone. Both IPv4 and IPv6 are supported.

8.8.1 Example

Automatic forward records

```
mod-synthrecord:
  - id: test1
    type: forward
    prefix: dynamic-
    ttl: 400
    network: 2620:0:b61::/52

zone:
  - domain: test.
    file: test.zone # Must exist
    module: mod-synthrecord/test1
```

Result:

```
$ kdig AAAA dynamic-2620-0000-0b61-0100-0000-0000-0000-0001.test.
...
;; QUESTION SECTION:
;; dynamic-2620-0000-0b61-0100-0000-0000-0000-0001.test. IN AAAA

;; ANSWER SECTION:
dynamic-2620-0000-0b61-0100-0000-0000-0000-0001.test. 400 IN AAAA 2620:0:b61:100::1
```

You can also have CNAME aliases to the dynamic records, which are going to be further resolved:

```
$ kdig AAAA alias.test.
...
;; QUESTION SECTION:
;; alias.test. IN AAAA

;; ANSWER SECTION:
alias.test. 3600 IN CNAME dynamic-2620-0000-0b61-0100-0000-0000-0000-0002.test.
dynamic-2620-0000-0b61-0100-0000-0000-0000-0002.test. 400 IN AAAA 2620:0:b61:100::2
```

Automatic reverse records

```
mod-synthrecord:
  - id: test2
    type: reverse
    prefix: dynamic-
    origin: test
    ttl: 400
    network: 2620:0:b61::/52

zone:
  - domain: 1.6.b.0.0.0.0.0.0.2.6.2.ip6.arpa.
    file: 1.6.b.0.0.0.0.0.0.2.6.2.ip6.arpa.zone # Must exist
    module: mod-synthrecord/test2
```

Result:

```
$ kdig -x 2620:0:b61::1
...
;; QUESTION SECTION:
;; 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.6.b.0.0.0.0.0.2.6.2.ip6.arpa. IN PTR

;; ANSWER SECTION:
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.6.b.0.0.0.0.0.2.6.2.ip6.arpa. 400 IN␣
↪PTR
                                dynamic-2620-0000-0b61-0000-0000-0000-0000-0001.
↪test.
```

8.8.2 Module reference

```
mod-synthrecord:
  - id: STR
    type: forward | reverse
    prefix: STR
    origin: DNAME
    ttl: INT
    network: ADDR[/INT] | ADDR-ADDR
```

id

A module identifier.

type

The type of generated records.

Possible values:

- `forward` – Forward records
- `reverse` – Reverse records

Required

prefix

A record owner prefix.

Note: The value doesn't allow dots, address parts in the synthetic names are separated with a dash.

Default: empty

origin

A zone origin (only valid for the *reverse type*).

Required

ttl

Time to live of the generated records.

Default: 3600

network

An IP address, a network subnet, or a network range the query must match.

Required

8.9 `whoami` — Whoami response

The module synthesizes an A or AAAA record containing the query source IP address, at the apex of the zone being served. It makes sure to allow Knot DNS to generate cacheable negative responses, and to allow fallback to extra records defined in the underlying zone file. The TTL of the synthesized record is copied from the TTL of the SOA record in the zone file.

Because a DNS query for type A or AAAA has nothing to do with whether the query occurs over IPv4 or IPv6, this module requires a special zone configuration to support both address families. For A queries, the underlying zone must have a set of nameservers that only have IPv4 addresses, and for AAAA queries, the underlying zone must have a set of nameservers that only have IPv6 addresses.

8.9.1 Example

To enable this module, you need to add something like the following to the Knot DNS configuration file:

```
zone:
  - domain: whoami.domain.example
    file: "/path/to/whoami.domain.example"
    module: mod-whoami

zone:
  - domain: whoami6.domain.example
    file: "/path/to/whoami6.domain.example"
    module: mod-whoami
```

The whoami.domain.example zone file example:

```
$TTL 1

@       SOA     (
                        whoami.domain.example.          ; MNAME
                        hostmaster.domain.example.      ; RNAME
                        2016051300                      ; SERIAL
                        86400                           ; REFRESH
                        86400                           ; RETRY
                        86400                           ; EXPIRE
                        1                               ; MINIMUM
                )

$TTL 86400

@       NS      ns1.whoami.domain.example.
@       NS      ns2.whoami.domain.example.
@       NS      ns3.whoami.domain.example.
@       NS      ns4.whoami.domain.example.

ns1     A       198.51.100.53
ns2     A       192.0.2.53
ns3     A       203.0.113.53
ns4     A       198.19.123.53
```

The whoami6.domain.example zone file example:

```
$TTL 1

@       SOA     (
                        whoami6.domain.example.         ; MNAME
                        hostmaster.domain.example.      ; RNAME
                        2016051300                      ; SERIAL
                        86400                           ; REFRESH
                        86400                           ; RETRY
                        86400                           ; EXPIRE
                        1                               ; MINIMUM
                )

$TTL 86400

@       NS      ns1.whoami6.domain.example.
@       NS      ns2.whoami6.domain.example.
@       NS      ns3.whoami6.domain.example.
@       NS      ns4.whoami6.domain.example.

ns1     AAAA    2001:db8:100::53
ns2     AAAA    2001:db8:200::53
```

```
ns3     AAAA    2001:db8:300::53
ns4     AAAA    2001:db8:400::53
```

The parent domain would then delegate whoami.domain.example to ns[1-4].whoami.domain.example and whoami6.domain.example to ns[1-4].whoami6.domain.example, and include the corresponding A-only or AAAA-only glue records.

Note: This module is not configurable.

CHAPTER

NINE

UTILITIES

Knot DNS comes with a few DNS client utilities and a few utilities to control the server. This section collects manual pages for all provided binaries:

9.1 kdig – Advanced DNS lookup utility

9.1.1 Synopsis

kdig [*common-settings*] [*query* [*settings*]]...

kdig -h

9.1.2 Description

This utility sends one or more DNS queries to a nameserver. Each query can have individual *settings*, or it can be specified globally via *common-settings*, which must precede *query* specification.

Parameters

query *name* | **-q** *name* | **-x** *address* | **-G** *tapfile*

common-settings, ***settings*** [*query_class*] [*query_type*] [*@server*]... [*options*]

name Is a domain name that is to be looked up.

server Is a domain name or an IPv4 or IPv6 address of the nameserver to send a query to. An additional port can be specified using address:port ([address]:port for IPv6 address), address@port, or address#port notation. If no server is specified, the servers from `/etc/resolv.conf` are used.

If no arguments are provided, **kdig** sends NS query for the root zone.

Query classes

A *query_class* can be either a DNS class name (IN, CH) or generic class specification **CLASS***XXXXX* where *XXXXX* is a corresponding decimal class number. The default query class is IN.

Query types

A *query_type* can be either a DNS resource record type (A, AAAA, NS, SOA, DNSKEY, ANY, etc.) or one of the following:

TYPE*XXXXX* Generic query type specification where *XXXXX* is a corresponding decimal type number.

AXFR Full zone transfer request.

IXFR=*serial* Incremental zone transfer request for specified starting SOA serial number.

NOTIFY=*serial* Notify message with a SOA serial hint specified.

NOTIFY Notify message with a SOA serial hint unspecified.

The default query type is A.

Options

-4 Use the IPv4 protocol only.

-6 Use the IPv6 protocol only.

-b *address* Set the source IP address of the query to *address*. The address must be a valid address for local interface or :: or 0.0.0.0. An optional port can be specified in the same format as the *server* value.

-c *class* An explicit *query_class* specification. See possible values above.

-d Enable debug messages.

-h, –help Print the program help.

-k *keyfile* Use the TSIG key stored in a file *keyfile* to authenticate the request. The file must contain the key in the same format as accepted by the **-y** option.

-p *port* Set the nameserver port number or service name to send a query to. The default port is 53.

-q *name* Set the query name. An explicit variant of *name* specification.

-t *type* An explicit *query_type* specification. See possible values above.

-V, –version Print the program version.

-x *address* Send a reverse (PTR) query for IPv4 or IPv6 *address*. The correct name, class and type is set automatically.

-y [*alg*:]*name*:*key* Use the TSIG key named *name* to authenticate the request. The *alg* part specifies the algorithm (the default is hmac-sha256) and *key* specifies the shared secret encoded in Base64.

-E *tapfile* Export a dnstap trace of the query and response messages received to the file *tapfile*.

-G *tapfile* Generate message output from a previously saved dnstap file *tapfile*.

+[no]multiline Wrap long records to more lines and improve human readability.

+[no]short Show record data only.

+[no]generic Use the generic representation format when printing resource record types and data.

+[no]crypto Display the DNSSEC keys and signatures values in hexdump, instead of omitting them.

+[no]aaflag Set the AA flag.

+[no]tcflag Set the TC flag.

+[no]rdflag Set the RD flag.

+[no]recurse Same as **+[no]rdflag**

+[no]raflag Set the RA flag.

+[no]zflag Set the zero flag bit.

+[no]adflag Set the AD flag.

+[no]cdflag Set the CD flag.

+[no]dnssec Set the DO flag.

+[no]all Show all packet sections.

+[no]qr Show the query packet.

+[no]header Show the packet header.

+[no]opt Show the EDNS pseudosection.

+[no]question Show the question section.

+[no]answer Show the answer section.

+[no]authority Show the authority section.

+[no]additional Show the additional section.

+[no]tsig Show the TSIG pseudosection.

+[no]stats Show trailing packet statistics.

+[no]class Show the DNS class.

+[no]ttl Show the TTL value.

+[no]tcp Use the TCP protocol (default is UDP for standard query and TCP for AXFR/IXFR).

+[no]ignore Don't use TCP automatically if a truncated reply is received.

+[no]tls Use TLS with the Opportunistic privacy profile.

+[no]tls-ca[=*FILE*] Use TLS with the Out-Of-Band privacy profile, use a specified PEM file (default is system certificate storage if no argument is provided). Can be specified multiple times.

+[no]tls-pin=*BASE64* Use TLS with a pinned certificate check. The PIN must be a Base64 encoded SHA-256 hash of the X.509 SubjectPublicKeyInfo. Can be specified multiple times.

+[no]tls-hostname=*STR* Use TLS with a remote server hostname check.

+[no]nsid Request the nameserver identifier (NSID).

+[no]bufsize=*B* Set EDNS buffer size in bytes (default is 512 bytes).

+[no]padding[=*B*] Use EDNS(0) padding option to pad queries, optionally to a specific size. The default is to pad queries with a sensible amount when using +tls, and not to pad at all when queries are sent without TLS. With no argument (i.e., just +padding) pad every query with a sensible amount regardless of the use of TLS. With +nopadding, never pad.

+[no]alignment[=*B*] Align the query to B-byte-block message using the EDNS(0) padding option (default is no or 128 if no argument is specified).

+[no]subnet=*SUBN* Set EDNS(0) client subnet SUBN=addr/prefix.

+[no]edns[=*N*] Use EDNS version (default is 0).

+[no]time=*T* Set the wait-for-reply interval in seconds (default is 5 seconds). This timeout applies to each query attempt.

+[no]retry=*N* Set the number (>=0) of UDP retries (default is 2). This doesn't apply to AXFR/IXFR.

+noidn Disable the IDN transformation to ASCII and vice versa. IDNA2003 support depends on libidn availability during project building!

9.1.3 Notes

Options **-k** and **-y** can not be used simultaneously.

Dnssec-keygen keyfile format is not supported. Use *keymgr(8)* instead.

9.1.4 Examples

1. Get A records for example.com:

```
$ kdig example.com A
```

2. Perform AXFR for zone example.com from the server 192.0.2.1:

```
$ kdig example.com -t AXFR @192.0.2.1
```

3. Get A records for example.com from 192.0.2.1 and reverse lookup for address 2001:DB8::1 from 192.0.2.2. Both using the TCP protocol:

```
$ kdig +tcp example.com -t A @192.0.2.1 -x 2001:DB8::1 @192.0.2.2
```

4. Get SOA record for example.com, use TLS, use system certificates, check for specified hostname, check for certificate pin, and print additional debug info:

```
$ kdig -d @185.49.141.38 +tls-ca +tls-host=getdnsapi.net \
  +tls-pin=foxZRnIh9gZpWnl+zEiKa0EJ2rdCGroMWm02gaxSc9S= soa example.com
```

9.1.5 Files

`/etc/resolv.conf`

9.1.6 See Also

khost(1), *knsupdate(1)*, *keymgr(8)*.

9.2 keymgr – Key management utility

9.2.1 Synopsis

keymgr *basic_option* [*parameters...*]

keymgr [*config_option config_storage*] *zone command argument...*

9.2.2 Description

The **keymgr** utility serves for manual key management in Knot DNS server.

Functions for DNSSEC keys and KASP (Key And Signature Policy) management are provided.

The DNSSEC and KASP configuration is stored in a so called KASP database. The database is backed by LMDB.

Basic options

-h, –help Print the program help.

-V, –version Print the program version.

-t tsig_name [*tsig_algorithm*] [*tsig_bits*] Generates TSIG key. TSIG algorithm can be specified by string (default: hmac-sha256), bit length of the key by number (default: optimal length given by algorithm).

Config options

-c Use specified Knot DNS configuration file path.

-C Use specified Knot DNS configuration database path. The default configuration database, if exists, has a preference to the default configuration file.

-d Use specified KASP database path and default configuration.

Commands

list Prints the list of key IDs and parameters of keys belonging to the zone.

generate [*arguments*...] Generates new DNSSEC key and stores it in KASP database. Prints the key ID. This action takes some number of arguments (see below). Values for unspecified arguments are taken from corresponding policy (if *-c* or *-C* options used) or from Knot policy defaults.

import-bind *BIND_key_file* Imports a BIND-style key into KASP database (converting it to PEM format). Takes one argument: path to BIND key file (private or public, but both MUST exist).

import-pem *PEM_file* [*arguments*...] Imports a DNSSEC key from PEM file. The key parameters (same as for the generate action) need to be specified (mainly algorithm, timers...) because they are not contained in the PEM format.

set *key_spec* [*arguments*...] Changes a timing argument of an existing key to a new timestamp. *Key_spec* is either the key tag or a prefix of the key ID; *arguments* are like for **generate**, but just the timing-related ones.

ds [*key_spec*] Generate DS record (all digest algorithms together) for specified key. *Key_spec* is like for **set**, if unspecified, all KSKs are used.

dnskey [*key_spec*] Generate DNSKEY record for specified key. *Key_spec* is like for **ds**, if unspecified, all KSKs are used.

delete *key_spec* Remove the specified key from zone. If the key was not shared, it is also deleted from keystore.

share *key_ID* Import a key (specified by full key ID) from another zone as shared. After this, the key is owned by both zones equally.

Generate arguments

Arguments are separated by space, each of them is in format 'name=value'.

algorithm Either an algorithm number (e.g. 14), or text name without dashes (e.g. ECDSAP384SHA384).

size Key length in bits.

ksk Either 'true' (KSK will be generated) or 'false' (ZSK will be generated).

created Timestamp of key creation.

publish Timestamp for key to be published.

ready Timestamp for key to be pre-activated and submitted (in case of KSK).

active Timestamp for key to be activated.

retire Timestamp for key to be de-activated.

remove Timestamp for key ot be deleted.

Timestamps

0 Zero timestamp means infinite future.

UNIX_time Positive number of seconds since 1970.

YYYYMMDDHHMMSS Date and time in this format without any punctuation.

relative_timestamp The word "now" followed by sign (+, -), a number and a shortcut for time unit (y, mo, d, h, mi, (nothing = seconds)), e.g. now+1mi, now-2mo, now+10, now+0, now-1y, ...

9.2.3 Examples

1. Generate new TSIG key:

```
$ keymgr -t my_name hmac-sha384
```

2. Generate new DNSSEC key:

```
$ keymgr example.com. generate algorithm=ECDSAP256SHA256 size=256 \
  ksk=true created=1488034625 publish=20170223205611 retire=now+10mo␣
↪remove=now+1y
```

3. Import a DNSSEC key from BIND:

```
$ keymgr example.com. import-bind ~/bind/Kharbinge4d5.+007+63089.key
```

4. Configure key timing:

```
$ keymgr example.com. set 4208 active=now+2mi retire=now+4mi remove=now+5mi
```

5. Share a KSK from another zone:

```
$ keymgr example.com. share e687cf927029e9db7184d2ece6d663f5d1e5b0e9
```

9.2.4 See Also

RFC 6781 - DNSSEC Operational Practices. RFC 7583 - DNSSEC Key Rollover Timing Considerations.

knot.conf(5), *knotc(8)*, *knotd(8)*.

9.3 pykeymgr – Key management utility

9.3.1 Synopsis

pykeymgr [*global-options*] [*command...*] [*arguments...*]

9.3.2 Description

The **pykeymgr** utility serves for key management in Knot DNS server.

Functions for DNSSEC keys and KASP (Key And Signature Policy) management are provided.

The DNSSEC and KASP configuration is stored in a so called KASP database. The databse is backed by LMDB.

The utility requires installed python LMDB module, installed e.g. by:

```
$ pip install lmdb
```

Global options

-f, –force Skip some of consistency checks and continue with performed action with a warning.

-h, –help Print the program help.

Main commands

-i, –import *KASP_db_dir* Import the legacy JSON-format KASP database into the current LMDB-backed one. (You can import multiple databases at once by repeating this option.)

Parameters

KASP_db_dir A path to the KASP db. It is the directory where *data.mdb* and *lock.mdb* files are usually stored as well as legacy JSON configuration and *keys* subdirectory containing PEM files.

9.3.3 Examples

1. Import legacy JSON-based KASP db from Knot 2.4.x after upgrade:

```
$ pykemgr -i ${knot_data_dir}/keys
```

9.3.4 See Also

RFC 6781 - DNSSEC Operational Practices.

knot.conf(5), *knotc(8)*, *knotd(8)*.

9.4 khost – Simple DNS lookup utility

9.4.1 Synopsis

khost [*options*] *name* [*server*]

9.4.2 Description

This utility sends a DNS query for the *name* to the *server* and prints a reply in more user-readable form. For more advanced DNS queries use **kdig** instead.

Parameters

name Is a domain name that is to be looked up. If the *name* is IPv4 or IPv6 address the PTR query type is used.

server Is a name or an address of the nameserver to send a query to. The address can be specified using [address]:port notation. If no server is specified, the servers from `/etc/resolv.conf` are used.

If no arguments are provided, **khost** prints a short help.

Options

-4 Use the IPv4 protocol only.

-6 Use the IPv6 protocol only.

-a Send ANY query with verbose mode.

-d Enable debug messages.

-h, –help Print the program help.

-r Disable recursion.

-T Use the TCP protocol.

-v Enable verbose output.

-V, –version Print the program version.

-w Wait forever for the reply.

-c *class* Set the query class (e.g. CH, CLASS4). The default class is IN.

-t *type* Set the query type (e.g. NS, IXFR=12345, TYPE65535). The default is to send 3 queries (A, AAAA and MX).

-R *retries* The number (>=0) of UDP retries to query a nameserver. The default is 1.

-W *wait* The time to wait for a reply in seconds. This timeout applies to each query try. The default is 2 seconds.

9.4.3 Examples

1. Get the A, AAAA and MX records for example.com:

```
$ khost example.com
```

2. Get the reverse record for address 192.0.2.1:

```
$ khost 192.0.2.1
```

3. Perform a verbose zone transfer for zone example.com:

```
$ khost -t AXFR -v example.com
```

9.4.4 Files

`/etc/resolv.conf`

9.4.5 See Also

kdig(1), *knsupdate(1)*.

9.5 kjournalprint – Knot DNS journal print utility

9.5.1 Synopsis

kjournalprint [*options*] *journal_db zone_name*

9.5.2 Description

The program prints zone history stored in a journal database. As default, changes are colored for terminal.

Options

-l, –limit *limit* Limits the number of displayed changes.

-d, –debug Debug mode brief output.

-n, –no-color Removes changes coloring.

-z, –zone-list Instead of reading jurnal, display the list of zones in the DB. (*zone_name* not needed)

-h, –help Print the program help.

-V, –version Print the program version.

Parameters

journal_db A path to the journal database.

zone_name A name of the zone to print the history for.

9.5.3 Examples

Last (most recent) 5 changes without colors:

```
$ kjournalprint -nl 5 /var/lib/knot/journal example.com.
```

9.5.4 See Also

knotd(8), *knot.conf(5)*.

9.6 knotc – Knot DNS control utility

9.6.1 Synopsis

knotc [*parameters*] *action* [*action_args*]

9.6.2 Description

If no *action* is specified, the program is executed in interactive mode.

Parameters

-c, –config *file* Use a textual configuration file (default is `@config_dir@/knot.conf`).

-C, –confdb *directory* Use a binary configuration database directory (default is `@storage_dir@/confdb`). The default configuration database, if exists, has a preference to the default configuration file.

-s, –socket *path* Use a control UNIX socket path (default is `@run_dir@/knot.sock`).

-t, –timeout *seconds* Use a control timeout in seconds. Set 0 for infinity (default is 5).

-f, –force Forced operation. Overrides some checks.

-v, –verbose Enable debug output.

-h, –help Print the program help.

-V, –version Print the program version.

Actions

status [*detail*] Check if the server is running. Details are **version** for the running server version, **workers** for the numbers of worker threads, or **configure** for the configure summary.

stop Stop the server if running.

reload Reload the server configuration and modified zone files. All open zone transactions will be aborted!

stats [*module*[*.counter*]] Show global statistics counter(s). To print also counters with value 0, use force option.

zone-status *zone* [*filter*] Show the zone status. Filters are **+role**, **+serial**, **+transaction**, **+events**, and **+freeze**.

zone-check [*zone*...] Test if the server can load the zone. Semantic checks are executed if enabled in the configuration. (*)

zone-memstats [*zone*...] Estimate memory use for the zone. (*)

zone-reload [*zone*...] Trigger a zone reload from a disk without checking its modification time. For slave zone, the refresh from a master server is scheduled; for master zone, the notification of slave servers is scheduled. An open zone transaction will be aborted!

zone-refresh [*zone*...] Trigger a check for the zone serial on the zone's master. If the master has a newer zone, a transfer is scheduled. This command is valid for slave zones.

zone-retransfer [*zone*...] Trigger a zone transfer from the zone's master. The server doesn't check the serial of the master's zone. This command is valid for slave zones.

zone-flush [*zone*...] [+outdir *directory*] Trigger a zone journal flush into the zone file. If output dir is specified, instead of flushing the zonefile, the zone is dumped to a file in the specified directory.

zone-sign [*zone*...] Trigger a DNSSEC re-sign of the zone. Existing signatures will be dropped. This command is valid for zones with automatic DNSSEC signing.

zone-read *zone* [*owner* [*type*]] Get zone data that are currently being presented.

zone-begin *zone*... Begin a zone transaction.

zone-commit *zone*... Commit the zone transaction. All changes are applied to the zone.

zone-abort *zone*... Abort the zone transaction. All changes are discarded.

zone-diff *zone* Get zone changes within the transaction.

zone-get *zone* [*owner* [*type*]] Get zone data within the transaction.

zone-set *zone owner* [*ttl*] *type rdata* Add zone record within the transaction. The first record in a rrset requires a ttl value specified.

zone-unset *zone owner* [*type* [*rdata*]] Remove zone data within the transaction.

zone-purge *zone*... [*filter*...] Purge zone data, file, journal, timers, and kaspdb. Filters are **+expire**, **+timers**, **+zonefile**, **+journal**, and **+kaspdb**.

zone-stats *zone* [*module*[*.counter*]] Show zone statistics counter(s). To print also counters with value 0, use force option.

zone-freeze [*zone*...] Temporarily postpone zone-changing events (load, refresh, update, flush, and DNSSEC signing).

zone-thaw [*zone*...] Dismiss zone freeze.

zone-ksk-submitted ***zone*** Use when the zone's KSK rollover is in submittion phase. By calling this command the user confirms manually that the parent zone contains DS record for the new KSK in submission phase and the old KSK can be retired.

conf-init Initialize the configuration database. (*)

conf-check Check the server configuration. (*)

conf-import ***filename*** Import a configuration file into the configuration database. Ensure the server is not using the configuration database! (*)

conf-export ***filename*** Export the configuration database into a config file. (*)

conf-list [*item*] List the configuration database sections or section items.

conf-read [*item*] Read the item from the active configuration database.

conf-begin Begin a writing configuration database transaction. Only one transaction can be opened at a time.

conf-commit Commit the configuration database transaction.

conf-abort Rollback the configuration database transaction.

conf-diff [*item*] Get the item difference in the transaction.

conf-get [*item*] Get the item data from the transaction.

conf-set ***item*** **[*data*...]** Set the item data in the transaction.

conf-unset [*item*] [*data*...] Unset the item data in the transaction.

Note

Empty or **–** *zone* parameter means all zones or all zones with a transaction.

Use @ *owner* to denote the zone name.

Type *item* parameter in the form of *section*[**[***id***]**][**.***name*].

(*) indicates a local operation which requires a configuration.

Interactive mode

The utility provides interactive mode with basic line editing functionality, command completion, and command history.

Interactive mode behavior can be customized in *~/.editrc*. Refer to `editrc(5)` for details.

Command history is saved in *~/.knotc_history*.

9.6.3 Examples

Reload the whole server configuration

```
$ knotc reload
```

Flush the example.com and example.org zones

```
$ knotc zone-flush example.com example.org
```

Get the current server configuration

```
$ knotc conf-read server
```

Get the list of the current zones

```
$ knotc conf-read zone.domain
```

Get the master remotes for the example.com zone

```
$ knotc conf-read 'zone[example.com].master'
```

Add example.org zone with a zonefile location

```
$ knotc conf-begin
$ knotc conf-set 'zone[example.org]'
$ knotc conf-set 'zone[example.org].file' '/var/zones/example.org.zone'
$ knotc conf-commit
```

Get the SOA record for each configured zone

```
$ knotc zone-read -- @ SOA
```

9.6.4 See Also

knotd(8), *knot.conf(5)*, *editrc(5)*.

9.7 knotd – Knot DNS server daemon

9.7.1 Synopsis

knotd [*parameters*]

9.7.2 Description

Parameters

-c, –config *file* Use a textual configuration file (default is `@config_dir@/knot.conf`).

-C, –confdb *directory* Use a binary configuration database directory (default is `@storage_dir@/confdb`). The default configuration database, if exists, has a preference to the default configuration file.

-s, –socket *path* Use a remote control UNIX socket path (default is `@run_dir@/knot.sock`).

-d, –daemonize [*directory*] Run the server as a daemon. New root directory may be specified (default is `/`).

-v, –verbose Enable debug output.

-h, –help Print the program help.

-V, –version Print the program version.

9.7.3 See Also

knot.conf(5), *knotc(8)*, *keymgr(8)*, *kjournalprint(1)*.

9.8 knsec3hash – NSEC hash computation utility

9.8.1 Synopsis

knsec3hash *salt algorithm iterations name*

9.8.2 Description

This utility generates a NSEC3 hash for a given domain name and parameters of NSEC3 hash.

Parameters

salt Specifies a binary salt encoded as a hexadecimal string.

algorithm Specifies a hashing algorithm by number. Currently, the only supported algorithm is SHA-1 (number 1).

iterations Specifies the number of additional iterations of the hashing algorithm.

name Specifies the domain name to be hashed.

9.8.3 Examples

```
$ knsec3hash c01dcafe 1 10 knot-dns.cz
7PTVGE7QV67EM61ROS9238P5RAKR2DM7 (salt=c01dcafe, hash=1, iterations=10)
```

```
$ knsec3hash - 1 0 net
A1RT98BS5QGC9NFI51S9HCI47ULJG6JH (salt=-, hash=1, iterations=0)
```

9.8.4 See Also

RFC 5155 – DNS Security (DNSSEC) Hashed Authenticated Denial of Existence.

knotc(8), *knotd(8)*.

9.9 knsupdate – Dynamic DNS update utility

9.9.1 Synopsis

knsupdate [*options*] [*filename*]

9.9.2 Description

This utility sends Dynamic DNS update messages to a DNS server. Update content is read from a file (if the parameter *filename* is given) or from the standard input.

The format of updates is textual and is made up of commands. Every command is placed on the separate line of the input. Lines starting with a semicolon are comments and are not processed.

Options

-d Enable debug messages.

-h, –help Print the program help.

-k *keyfile* Use the TSIG key stored in a file *keyfile* to authenticate the request. The file should contain the key in the same format, which is accepted by the **-y** option.

-p *port* Set the port to use for connections to the server (if not explicitly specified in the update). The default is 53.

-r *retries* The number of retries for UDP requests. The default is 3.

-t *timeout* The total timeout (for all UDP update tries) of the update request in seconds. The default is 12. If set to zero, the timeout is infinite.

-v Use a TCP connection.

-V, –version Print the program version.

-y [*alg*:]*name*:*key* Use the TSIG key with a name *name* to authenticate the request. The *alg* part specifies the algorithm (the default is hmac-sha256) and *key* specifies the shared secret encoded in Base64.

Commands

server *name* [*port*] Specifies a receiving server of the dynamic update message. The *name* parameter can be either a host name or an IP address. If the *port* is not specified, the default port is used. The default port value can be controlled using the **-p** program option.

local *address* [*port*] Specifies outgoing *address* and *port*. If no local is specified, the address and port are set by the system automatically. The default port number is 0.

zone *name* Specifies that all updates are done within a zone *name*. If not used, the default zone is the root zone.

origin *name* Specifies fully qualified domain name suffix which is appended to non-fqd owners in update commands. The default origin is the root zone.

class *name* Sets *name* as the default class for all updates. If not used, the default class is IN.

ttl *value* Sets *value* as the default TTL (in seconds). If not used, the default value is 0.

key [*alg*:]*name key* Specifies the TSIG *key* named *name* to authenticate the request. An optional *alg* algorithm can be specified. This command has the same effect as the program option **-y**.

[prereq] nxdomain *name* Adds a prerequisite for a non-existing record owned by *name*.

[prereq] yxdomain *name* Adds a prerequisite for an existing record owned by *name*.

[prereq] nxrrset *name* [*class*] *type* Adds a prerequisite for a non-existing record of the *type* owned by *name*. Internet *class* is expected.

[prereq] yxrrset *name* [*class*] *type* [*data*] Adds a prerequisite for an existing record of the *type* owned by *name* with optional *data*. Internet *class* is expected.

[update] add *name* [*ttl*] [*class*] *type data* Adds a request to add a new resource record into the zone. Please note that if the *name* is not fully qualified domain name, the current origin name is appended to it.

[update] del[ete] *name* [*ttl*] [*class*] [*type*] [*data*] Adds a request to remove all (or matching *class*, *type* or *data*) resource records from the zone. There is the same requirement for the *name* parameter as in **update add** command. The *ttl* item is ignored.

show Displays current content of the update message.

send Sends the current update message and cleans the list of updates.

answer Displays the last answer from the server.

debug Enable debugging. This command has the same meaning as the **-d** program option.

quit Quit the program.

9.9.3 Notes

Options **-k** and **-y** can not be used simultaneously.

Dnssec-keygen keyfile format is not supported. Use *keymgr(8)* instead.

Zone name/server guessing is not supported if the zone name/server is not specified.

Empty line doesn't send the update.

9.9.4 Examples

1. Send one update of the zone example.com to the server 192.168.1.1. The update contains two new records:

```
$ knsupdate
> server 192.168.1.1
> zone example.com.
> origin example.com.
> ttl 3600
> add test1.example.com. 7200 A 192.168.2.2
> add test2 TXT "hello"
> show
> send
> answer
> quit
```

9.9.5 See Also

kdig(1), *khost(1)*, *keymgr(8)*.

9.10 kzonecheck – Knot DNS zone file checking tool

9.10.1 Synopsis

kzonecheck [*options*] *filename*

9.10.2 Description

The utility checks zone file syntax and runs semantic checks on the zone content. The executed checks are the same as the checks run by the Knot DNS server.

Please, refer to the `semantic-checks` configuration option in *knot.conf(5)* for the full list of available semantic checks.

Options

-o, –origin *origin* Zone origin. If not specified, the origin is determined from the file name (possibly removing the `.zone` suffix).

-v, –verbose Enable debug output.

-h, –help Print the program help.

-V, –version Print the program version.

9.10.3 See Also

knotd(8), *knot.conf(5)*.

CHAPTER

TEN

MIGRATION

10.1 Upgrade 2.4.x to 2.5.x

This chapter describes some steps necessary after upgrading Knot DNS from version 2.4.x to 2.5.x.

10.1.1 Building changes

The `--enable-dnstap` configure option now enables the dnstap support in kdig only! To build the dnstap query module, `--with-module-dnstap` have to be used.

Since Knot DNS version 2.5.0 each query module can be configured to be:

- disabled: `--with-module-`MODULE_NAME`=no`
- embedded: `--with-module-`MODULE_NAME`=yes`
- external: `--with-module-`MODULE_NAME`=shared` (excluding `dnsproxy` and `onlinesign`)

The `--with-timer-mapsize` configure option was replaced with the runtime *max-timer-db-size* configuration option.

10.1.2 KASP DB migration

Knot DNS version 2.4.x and earlier uses JSON files to store DNSSEC keys metadata, one for each zone. 2.5.x versions store those in binary format in a LMDB, all zones together. The migration is possible with `pykeymgr` script:

```
$ pykeymgr -i path/to/keydir
```

The path to KASP DB directory is configuration-dependent, usually it is the `keys` subdirectory in the zone storage.

In rare installations, the JSON files might be spread across more directories. In such case, it is necessary to put them together into one directory and migrate at once.

10.1.3 Configuration changes

It is no longer possible to configure KASP DB per zone or in a non-default template. Ensure just one common KASP DB configuration in the default template.

As Knot DNS version 2.5.0 brings dynamically loaded modules, some modules were renamed for technical reasons. So it is necessary to rename all occurrences (module section names and references from zones or templates) of the following module names in the configuration:

```
mod-online-sign -> mod-onlinesign

mod-synth-record -> mod-synthrecord
```

10.2 Knot DNS for BIND users

10.2.1 Automatic DNSSEC signing

Migrating automatically signed zones from BIND to Knot DNS requires copying up-to-date zone files from BIND, importing existing private keys, and updating server configuration:

1. To obtain current content of the zone which is being migrated, request BIND to flush the zone into the zone file: `rndc flush example.com`.

   Note: If dynamic updates (DDNS) are enabled for the given zone, you might need to freeze the zone before flushing it. That can be done similarly:

   ```
   $ rndc freeze example.com
   ```

2. Copy the fresh zone file into the zones *storage* directory of Knot DNS.

3. Import all existing zone keys into the KASP database. Make sure that all the keys were imported correctly:

   ```
   $ keymgr example.com. import-bind path/to/Kexample.com.+013+11111
   $ keymgr example.com. import-bind path/to/Kexample.com.+013+22222
   $ ...
   $ keymgr example.com. list
   ```

   Note: The server can be run under a dedicated user account, usually `knot`. As the server requires read-write access to the KASP database, the permissions must be set correctly. This can be achieved for instance by executing all KASP database management commands under sudo:

   ```
   $ sudo -u knot keymgr ...
   ```

4. Follow *Automatic DNSSEC signing* steps to configure DNSSEC signing.

CHAPTER

ELEVEN

APPENDICES

11.1 Compatible PKCS #11 Devices

This section has informative character. Knot DNS has been tested with several devices which claim to support PKCS #11 interface. The following table indicates which algorithms and operations have been observed to work. Please notice minimal GnuTLS library version required for particular algorithm support.

	Key generate	Key import	ECDSA 256-bit	ECDSA 384-bit	RSA 1024-bit	RSA 2048-bit	RSA 4096-bit	DSA 512-bit	DSA 1024-bit
Feitian ePass 2003	**yes**	no	no	no	**yes**	**yes**	no	no	no
SafeNet Network HSM (Luna SA 4)	**yes**	no	no	no	**yes**	**yes**	**yes**	no	no
SoftHSM 2.0	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**
Trustway Proteccio NetHSM	**yes**	ECDSA only	**yes**	**yes**	**yes**	**yes**	**yes**	no	no

The following table summarizes supported DNSSEC algorithm numbers and minimal GnuTLS library version required. Any algorithm may work with older library, however the supported operations may be limited (e.g. private key import).

	Numbers	GnuTLS version
ECDSA	13, 14	3.4.8 or newer
RSA	5, 7, 8, 10	3.4.6 or newer
DSA	3, 6	3.4.10 or newer

R