slimDNS

Release v1.0.0rc1

Jul 19, 2020

Programming Guide

slimDNS is a simple, minimal and flexible DNS server.
It doesn't require external dependencies and work off Python 3.8 builtins.

Here's a demo domain using minimal setup:

```
import slimDNS

dns = slimDNS.server(slimDNS.UDP)

dns.run()
@dns.records
def records(server):
    return {
        "hvornum.se" : {
              "A" : {"target" : "46.21.102.81", "ttl" : 60},
              "SOA" : {"target" : "hvornum.se", "ttl" : 60},
              "NS" : {"target" : "hvornum.se", "ttl" : 60, "priority" : 10}
        }
    }
```

Some of the features of slimDNS are:

- **No external dependencies or installation requirements.** Runs without any external requirements or installation processes.
- **Single threaded.** slimDNS takes advantage of *select.epoll() (select.select() on Windows)* to achieve blazing speeds without threading the service. Threads are allowed and welcome, but the core code relies on using as few threads and overhead as possible.

CHAPTER 1

Installation

Note: These instructions apply to slimDNS .

slimDNS is a pure python library, so no special steps are required for installation. You can install it in a variety of ways described below though for your convenience.

1.1 Using *pip*

```
pip install slimDNS
```

1.2 Clone using *git*

```
git clone https://github.com/Torxed/slimDNS.git
```

But most likely you'll want to submodule this in a project. To do that, I would recommend not following master as it's actively developed. Any release/tag should be good enough for production.

```
cd project/dependencies
git submodule add -b v1.0 https://github.com/Torxed/slimDNS.git
```

Which would follow the stable release branch of *v1.0* where tests *should* be done before release.

1.3 Manually unpacking source

The source code archives *(including git)* includes examples. Archives are available on Github:

```
unzip slimDNS-x.x.x.zip
cd slimDNS-x.x.x
python examples/dns_server.py
```

CHAPTER 2

Configuration

Configuration is done by supplying slimDNS with a *dict* of records.
A complete example can be found under *Baisc setup config*.
But as a small example, this would run a DNS server:

Warning:

There's two startup-sensitive configuration options.
Those are *addr* and *port* to set the listening interface other than default *0.0.0.0:53*.

To delcare *addr* and *port* - you have to do it from the startup code:

```
import slimDNS

http = slimDNS.server(slimDNS.UDP, addr='127.0.0.1', port=8080)
http.run()
```

Trying to set it in the runtime configuration will fail, as the server has already setup the *socket.bind((addr, port))*

Note:

Also note that record definitions are done by the developers code that imported slimDNS.
It's there for up to the developer if the records should be stored on disk in a particular format or in the code itself.

Note:

All records can be changed in runtime without needing to reload the server.
The records can be modified in four different ways:

2.1 Baisc setup config

```
import slimDNS

dns = slimDNS.server(slimDNS.UDP)

@dns.records
def records(server):
    return {
        "example.com" : {
            "A" : {"target" : "264.30.198.2", "ttl" : 60},
            "SOA" : {"target" : "example.com", "ttl" : 60},
            "NS" : {"target" : "example.com", "ttl" : 60, "priority" : 10}
        },
        "nas.example.com" : {
            "A" : {"target" : "264.30.198.2", "type" : "A", "ttl" : 60}
        },
        "_matrix._tcp.riot.example.com" : {
            "SRV" : {"ttl" : 60, "priority" : 10, "port" : 8448, "target" : "nas.
↪example.com"}
        }
    }

dns.run()
```

Here, records are loaded after the server has started up.
All the records become active immediately weither or not they are valid/correct.

Warning:

The annotation *@dns.records* replaces any previous records.
This is there for only useful if you want to purge previous records and update with a new set.

2.2 Adding records

Records can be added one by one without replacing the previous ones with *add()*.

```
import slimDNS

dns = slimDNS.server(slimDNS.UDP)

dns.add('example.com', 'A', '264.30.198.1')

dns.run()
```

2.3 Removing records

Records can be deleted one by one without affecting other records via the *remove()*.

```
import slimDNS

dns = slimDNS.server(slimDNS.UDP)

dns.remove('example.com', 'A')

dns.run()
```

2.4 Updating records

Records can also be updated individually *update()*.

```
import slimDNS

dns = slimDNS.server(slimDNS.UDP)

dns.update('example.com', 'A', '264.30.198.5')

dns.run()
```

CHAPTER 3

Discord

There's a discord channel which is frequent by the contributors.

To join the server, head over to discord.com/slimDNS and join in. There's not many rules other than common sense and treat others with respect.

There's the *@Party Animals* role if you want notifications of new releases which is posted in the *#Release Party* channel. Another thing is the *@Contributors* role which you can get by writing *!verify* and verify that you're a contributor.

Hop in, I hope to see you there! :)

CHAPTER 4

Issue tracker

Issues whould be reported over at GitHub/issues.

General questions, enhancements and security issues can be reported over there too. For quick issues or if you need help, head over the to the Discord.

CHAPTER 5

A Record

The A record is a general purpose pointer to an IP or CNAME destination.
Arguably the most common DNS record you'll ever use.

To create such a record, assuming the main server instance is called *dns*, you'd simply do:

```
dns.add('hvornum.se', 'A', '192.168.10.1')
```

This will resolve all DNS request for the host *hvornum.se* to *192.168.10.1*.

CHAPTER 6

NS Record

The NS record is the Name Server record.
This points clients and servers towards a DNS server for a particular domain/name/record.

Note: This example assuming the main server instance is called *dns*.

```
dns.add('hvornum.se', 'NS', 'ns1.hvornum.se')
```

This creates a record that tells anyone asking, that the primary name server for *hvornum.se* is located at *ns1.hvornum.se*.

Note: There are other options as well, such as *ttl=3600*. But a basic record can be created with only these three options.

CHAPTER 7

SOA Record

The SOA record is the *Start of Authority* record.
This contains all the main information top DNS servers need to identify your DNS server as a authority for the asked domain.

Note: This example assuming the main server instance is called *dns*.

```
dns.add('hvornum.se', 'SOA', email='root@hvornum.se', target='ns1.hvornum.se')
```

This sets up a *SOA* record for the domain *hvornum.se*, where the primary DNS server (*target´) is 'ns1.hvornum.se*.
And the contact e-mail for the domain is *root@hvornum.se*.

CHAPTER 8

SRV Record

The SRV record is a service locator record.
This helps others identify certain services such as Active Directory, chat servers etc.

Note: This example assuming the main server instance is called *dns*.

```
dns.add('_chat._tcp.hvornum.se', 'SRV', target='chat.hvornum.se', port=8080,␣
↪priority=10)
```

This creates a *SRV* record that tells a chat application, that the primary host and port for the chat is located at *tcp://chat.hvornum.se:8080*.

CHAPTER 9

slimDNS.server

slimDNS.**server**(*mode=2*, **args*, ***kwargs*)

server() is essentially just a router. It creates a instance of a selected mode (either *TCP* or *UDP*). It also saves the instance in a shared instance variable for access later.

CHAPTER 10

slimDNS.TCP_SERVER

class slimDNS.**TCP_SERVER**(**args*, ***kwargs*)

Bases: object

add(*record*, *record_type*, *target*, *ttl=60*, ***kwargs*)

default_config()

Returns a simple but sane default configuration in case no one is given.

Returns {'addr' : '', 'port' : 53}

Return type dict

log(**args*, ***kwargs*)

A simple print wrapper, placeholder for more advanced logging in the future. Joins any **args* together and safely calls :func:'str' on each argument.

poll(*timeout=0.001*)

records(*f*, **args*, ***kwargs*)

remove(*record*, *record_type*)

run()

setup_socket()

update(*record*, *record_type*, ***kwargs*)

CHAPTER 11

slimDNS.UDP_SERVER

class slimDNS.**UDP_SERVER**(**args*, ***kwargs*)

Bases: slimDNS.lib.servers.TCP_SERVER

poll(*timeout=0.001*)

run()

setup_socket()

CHAPTER 12

slimDNS.DNS_RESPONSE

class slimDNS.**DNS_RESPONSE**(*DNS_FRAME*, *header_length*)

Bases: object

additionals

answers

assemble

authorities

queries

CHAPTER 13

slimDNS.QUERY

class slimDNS.**QUERY**(*RAW_FRAME*, *query_type*, *record*, *query_class='IN'*)

Bases: object

bytes

CHAPTER 14

slimDNS.ANSWER

class slimDNS.**ANSWER**(*FRAME*, *data*)

Bases: slimDNS.lib.data.DNS_FIELD

CHAPTER 15

slimDNS.ADDITIONAL

class slimDNS.**ADDITIONAL**(*FRAME*, *data*)

Bases: slimDNS.lib.data.DNS_FIELD

CHAPTER 16

slimDNS.Events

class slimDNS.**Events**

Bases: object

Events.<CONST> is a helper class to indicate which event is triggered. Events are passed up through the event chain deep from within slimDNS.

These events can be caught in your main *.poll()* loop, and react to different events.

CLIENT_DATA = 64

CLIENT_INVALID_DATA = 67

CLIENT_NO_QUERIES = 65

CLIENT_RESPONSE_DATA = 66

DATA_EVENTS = (66,)

NOT_YET_IMPLEMENTED = 0

SERVER_ACCEPT = 128

SERVER_CLOSE = 129

SERVER_RESTART = 2

convert()

Index