VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FRAMEWORK NA TESTOVÁNÍ DNS SERVERŮ

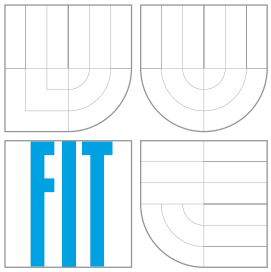
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN NOVÁK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

FRAMEWORK NA TESTOVÁNÍ DNS SERVERŮ

FRAMEWORK FOR DNS SERVER TESTING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN NOVÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR MATOUŠEK, Ph.D.

BRNO 2015

Abstrakt

Tato práce se zabývá úpravami frameworku určeného pro testování DNS serverů. Framework je vyvíjen sdružením NIC.CZ a slouží především pro testování jejich DNS serveru Knot DNS. Cílem této práce jsou úpravy frameworku, které umožní jednodušší testování za pomoci tohoto frameworku, jako například: podpora více implementací DNS serverů, paralelizace testování, prvky dummy server a box-in-the-middle, rozdělení na více komponent a celková úprava stávajícího frameworku. Úvod práce je věnován autoritativním DNS serverům a základům testování. Zbývající část práce se zabývá stavem dosavadního frameworku a stavem a testováním upraveného frameworku.

Abstract

This thesis deals with the modifications of the framework designed for DNS servers testing. Framework is developed by NIC.CZ association and is used primarily for testing the DNS server Knot DNS. The aim of this work are modifications of the framework that will allow simpler testing with this framework, such as: support for multiple implementations of DNS servers, parallel testing, components dummy server and box-in-the-middle, division into multiple components and overall modification of the existing framework. Introduction of thesis is dedicated to the authoritative DNS servers and to the foundations of testing. The remaining part of the thesis deals with the state of the existing framework and the state and testing of modified framework.

Klíčová slova

DNS, DNS framework, DNS server, testování, Python, dnspython, Knot DNS, BIND, NSD

Keywords

DNS, DNS framework, DNS server, testing, Python, dnspython, Knot DNS, BIND, NSD

Citace

Martin Novák: Framework na testování DNS serverů, diplomová práce, Brno, FIT VUT v Brně, 2015

Framework na testování DNS serverů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Petra Matouška, Ph.D. a ze strany technických konzultací pod vedením pana Ing. Jana Včeláka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Novák
25. května 2015

Poděkování

Rád by som poďakoval predovšetkým pánovi Ing. Petrovi Matouškovi, Ph.D. za vecné pripomienky, rady a informácie, ktoré pomohli vzniknúť tejto práci. Taktiež ďakujem pánovi Ing. Jánovi Včelákovi za poskytnutie mnohých rád a cenných informácií, ktoré boli nápomocné pri tvorbe implementácie.

Obsah

Kapitola 1

Úvod

Systém doménových mien (angl. *Domain Name System*, skrátene *DNS*) je neoddeliteľnou súčasťou dnešného internetu, tak ako ho poznáme. DNS je distribuovaná databáza umiestnená na špeciálnych serveroch – DNS serveroch, ktorej hlavnou úlohou je preklad doménových mien na IP adresy a opačne. Jedná sa o známu architektúru typu klient – server. Tento systém bol vyvinutý za účelom zjednodušenia práce užívateľom internetu a správcom serverov, ktoré sú zodpovedné za jednotlivé domény. Vďaka systému DNS si užívateľ nemusí pamätať mnoho zložitých číselných sekvencií – IP adries, ale stačí mu zapamätať si jednoduchšie doménové mená. Pre správcov sietí DNS systém znamená jednoduchšiu správu a udržateľnosť konzistencie databázy obsahujúcej okrem iného hlavne dvojice IP adries a mien a jednoduchšiu propagáciu týchto zmien do celej siete.

Nepretržitá dostupnosť DNS serverov je teda kritická pre funkciu internetu. Z tohoto dôvodu je potrebné všetky implementácie DNS serverov dôkladne otestovať v rôznych situáciách a znížiť tak riziko ich chýb alebo výpadkov. Platí, že čím „vyššie“ sa v architektúre systému DNS server nachádza, tým má jeho výpadok väčší dosah na celý systém. Preto by mali byť tieto servery najdôkladnejšie testované.

Náplňou tejto práce je úprava stávajúceho frameworku vyvíjaného združením CZ.NIC. Framework slúži predovšetkým na testovanie nimi vyvíjaného serveru Knot DNS a podporuje taktiež DNS server BIND. Framework je vytvorený v jazyku Python 3 a jeho úpravou by okrem iného mala byť aj podpora implementácií iných DNS serverov, zjednodušenie jeho používania a písania testov a podobne.

Text tejto práce oboznámi čitateľa so základnými princípmi autoritatívnych DNS serverov a ich testovaním. U čitateľa práce sa predpokladá základná znalosť DNS systému, jeho mechanizmov a zabezpečovacích prvkov. Začiatok práce (kap. 2) sa venuje všeobecným princípom autoritatívnych DNS serverov a ich použitím v systéme DNS. Nasledujú podkapitoly venované jednotlivým implementáciám DNS serverov, hlavne serveru Knot DNS. Samotné základy testovania DNS serverov sú rozobrané v kapitole 3. Stav stávajúceho frameworku, jeho štruktúra, použitie a výstupy spolu s požiadavkami na upravený framework sú popísané v kapitole 4.

Druhá polovica textu začína kapitolou 5, ktorá popisuje riešenia požiadaviek na úpravu. Nasleduje kapitola s príkladovými testami demonštrujúcimi funkčnosť upraveného frameworku (kap. 6) a kapitola popisujúca možné budúce úpravy frameworku (kap. 7).

Kapitola 2

Autoritatívne DNS servery

V nasledujúcom texte bude stručne popísané, čo je to autoritatívny DNS server a na čo sa používa. Nasleduje podkapitola 2.2 stručne popisujúca vybrané implementácie autoritatívnych DNS serverov, s ktorými sa čitateľ stretne v ďalšom texte, podkapitola 2.3 venovaná konfiguračným súborom, kde bude vysvetlené, na aký účel sa používajú, aký majú formát a aké sú medzi nimi rozdiely. Posledná podkapitola 2.4 sa venuje autoritatívnemu DNS serveru Knot DNS, konkrétne jednotlivým procesom, ktoré na servery Knot DNS prebiehajú (napríklad podpisovanie, prenos zón a podobne).

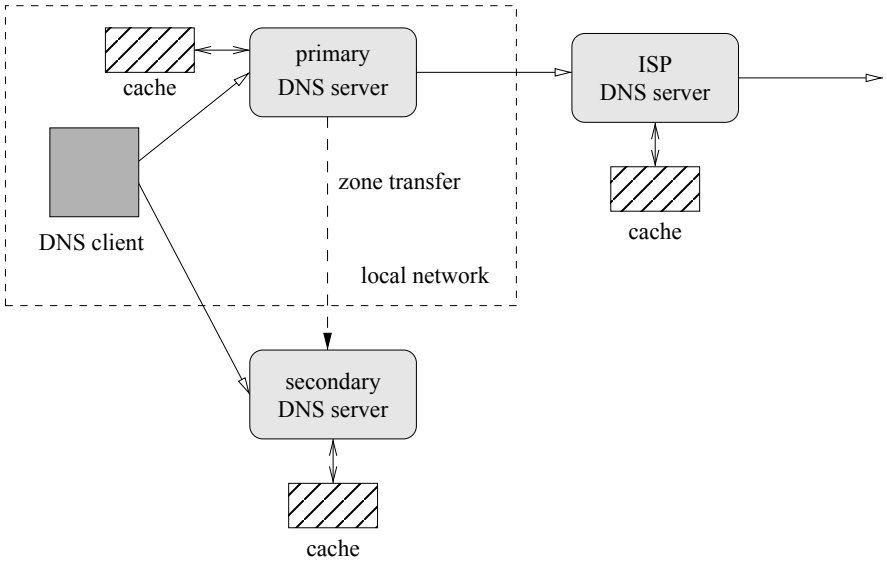
2.1 Čo je to autoritatívny DNS server?

Autoritatívne DNS servery sú DNS servery, ktoré vydávajú autoritatívne odpovede na dotazy ohľadom mien v ich zónach. Autoritatívny DNS server odpovedá iba na dotazy týkajúce sa doménových mien, ktoré boli špecificky nastavené administrátorom servera. DNS server môže byť tak isto nakonfigurovaný na vydávanie odpovedí aj pre iné zóny ako tie, pre ktoré vydáva autoritatívne odpovede. Môže fungovať v kombinovanom režime, kedy popri autoritatívnych odpovediach DNS server vydáva aj neautoritatívne odpovede, teda sa chová ako cache server[1].

Autoritatívne DNS servery delíme na dva typy:

- **Primárne DNS servery** (angl. *primary* alebo *master server*)
  DNS server, ktorý uchováva konečnú a aktuálnu verziu všetkých záznamov v ním spravovanej zóne. Záznamy sú uložené lokálne v súbore. Server poskytuje autoritatívne (presné a aktuálne) odpovede pre spravované zóny. Každá doména musí byť spravovaná presne jedným primárnym DNS serverom.
- **Sekundárne DNS servery** (angl. *secondary* alebo *slave server*)
  Sekundárny DNS server tvorí presnú repliku primárneho DNS servera, teda obsahuje presne tie isté zónové súbory ako primárny server. Sekundárny server je však tiež

[1] Cache server, v kontexte DNS serverov, je typ DNS serveru, ktorý uchováva po určitý čas DNS odpovede a tým zmenšuje zaťaženie ostatných DNS serverov a znižuje latenciu medzi položením dotazu a príchodom odpovede.



Obr. 2.1: Schéma príkladu zapojenia DNS serverov

autoritatívny pre danú doménu. Zónové súbory sú prenášané z primárneho na sekundárny server. Tento proces sa nazýva prenos zón (angl. *zone transfer*), popísaný v kapitole 2.4. Za aktuálnosť dát na sekundárnom servery je zodpovedný sám sekundárny server. Aktuálnosť zabezpečuje pravidelným prenosom zónových dát. Používa sa na rozloženie záťaže medzi DNS servermi v danej zóne a taktiež tvorí redundanciu služby prekladu doménových mien. Každá doména by mala obsahovať minimálne jeden sekundárny DNS server z vyššie uvedených dôvodov a ideálne by sa mali nachádzať na fyzicky rôznych sieťach.

Ich spôsob prepojenia môže byť realizovaný podobne ako na obrázku 2.1 prevzatého z [6]. Zapojenie obsahuje jeden primárny a jeden sekundárny DNS server a jeden ISP DNS server (server poskytovateľa internetu).

Systém doménových mien poskytuje mechanizmus na notifikáciu sekundárnych DNS serverov. Primárny server notifikuje všetky známe sekundárne DNS servery pre svoje zóny. Proces notifikácie je inicializovaný primárnym DNS serverom pri zmene zónových súborov, ktoré musia byť prenesené na sekundárne DNS servery. Proces notifikácie je bližšie popísaný v kapitole 2.4.

Obsah zónových súborov je spravovaný administrátorom alebo systémom dynamického DNS (kap. 2.4). Každé doménové meno je uvedené na jednom alebo viacerých autoritatívnych DNS serveroch. Plne kvalifikované doménové mená (angl. *Fully Qualified Domain Name*, skrátene *FQDN*) autoritatívnych serverov sú uvedené v zónových súboroch danej zóny, v záznamoch typu NS. V prípade, že nie sú tieto servery zároveň autoritatívnymi servermi pre rodičovskú zónu danej zóny, musí byť autoritatívny server rodičovskej zóny nakonfigurovaný tak, aby delegoval dotazy týkajúce sa danej zóny na dané autoritatívne servery.

Pri registrácii novej domény musí administrátor zóny poskytnúť zoznam autoritatívnych DNS serverov, ktoré budú poskytovať odpovede pre danú doménu. Regitrátor domén pridá tieto servery do záznamov svojich serverov tak, aby bolo možné sa k týmto novým domé-

novým menám (aut. serverom) dostať.

2.2 Vybrané autoritatívne DNS servery

V tejto kapitole sú popísané tie implementácie autoritatívnych DNS serverov, ktoré bude výsledný framework pre testovanie podporovať. Framework bude však ďalej rozšíriteľný (kap. 4.4) a bude možné pridať aj iné implementácie autoritatívnych DNS serverov.

Informácie v tejto a nasledujúcich kapitol boli čerpané z dokumentácií jednotlivých serverov, t.j. serveru BIND [9], serveru NSD [8] a serveru Knot [10].

V rámci výsledného frameworku budú podporované nasledovné implementácie autoritatívnych serverov:

- **BIND** (angl. *Berkeley Internet Name Domain*):
  Open-source[2] implementácia autoritatívneho DNS servera pre viacero operačných systémov. Vznikol na začiatku 80-tych rokov na univerzite v Kalifornii, dnes vyvíjaný spoločnosťou ISC (Internet Systems Consortium, Inc.). Dodnes patrí medzi najpoužívanejšie implementácie DNS serverov. Implementovaný je predovšetkým v jazyku C spolu s dodatočným kódom napísaným v jazykoch Python, Perl a Bourne shell. Podporuje viacero formátov zápisu zónových súborov od textových po binárne.
- **NSD**:
  Anglicky *Name Server Daemon* je open-source software, implementovaný v jazyku C a šírený pod licenciou BSD[3] vyvíjaný spoločnosťou NLnet Labs. Beží na operačných systémoch s Unixovým základom. Používa rovnaké formáty zónových súborov ako BIND, navyše podporuje kompilované zónové súbory pomocou nástroja `zonec`.
- **Knot DNS**:
  Knot DNS je open-source software šírený po GPL licenciou vyvíjaný združením CZ.NIC. Taktiež ako väčšina rozšírených implementácií DNS serverov, beží na operačných systémoch s Unixovým základom a je implementovaný v jazyku C. Podpora formátov zónových súborov je rovnaká ako u serveru BIND.

Prehľad vlastností vybraných DNS serverov je v tabuľke 2.1 získanej zo stránky wikipédie[4].

2.3 Konfigurácia serverov

Výstupom tejto práce je upravený framework, ktorý musí byť nezávislý na konkrétnej implementácii DNS serveru. Pre testovanie konkrétnych implementácií autoritatívnych DNS serverov však potrebujeme určité rozhranie, cez ktoré vieme tieto servery nakonfigurovať. K tomuto účelu slúžia konfiguračné a zónové súbory jednotlivých DNS serverov. Konfiguračné súbory však bohužiaľ nemajú určený žiadny štandard narozdiel od formátu zónových súborov, ktorý by určoval ich jednotný formát.

[2] Open-source software je software so zdrojovým kódom, ktorý je voľne prístupný pod určitou licenciou.
[3] `http://www.linfo.org/bsdlicense.html`
[4] `http://en.wikipedia.org/wiki/Comparison_of_DNS_server_software`

Server	Autoritatívny	Rekurzívny	Sekundárny	Cache	DNSSEC	TSIG	IPv6	Rozhranie	Split horizon
BIND	ano	ano	ano	ano	ano	ano	ano	web, prík. riadok	ano
NSD	ano	nie	ano	N/A	ano	ano	ano	prík. riadok	nie
Knot DNS	ano	nie	ano	N/A	ano	ano	ano	prík. riadok	nie

Tabuľka 2.1: Porovnanie niektorých vlastností vybraných DNS serverov

Nasledujúca podkapitola popisuje formát zónových súborov, ktorých formát je štandardizovaný. Ďalej bude popísané, ako možno jednotlivé servery konfigurovať, aké formáty konfiguračných súborov používajú a ktoré vlastnosti serverov možno v týchto nastaveniach meniť.

2.3.1 Zónové súbory

Systém DNS používa na uloženie dát o zónach tzv. zónové súbory. Ide o textové súbory, ktorých formát je definovaný v dokumentoch RFC 1035 [2] v sekcii 5 a RFC 1034 [1] v sekcii 3.6.1. Všetky validné zónové súbory teda musia dodržiavať predpísaný formát – syntax a sémantiku. Mnoho implementácií však tento textový súbor používa iba ako „štartovací“ formát a pri normálnej prevádzke používa zónové súbory transformované do rôznych úložísk ako databáz a podobne. Zónový súbor obsahuje záznamy (angl. *Resource Records* – skrátene *RR*) všetkých doménových mien asociovaných so zónou.

Formát zónového súboru podľa RFC 1035 [2] v sekcii 5 je na obrázku 2.2. Každý zónový súbor musí obsahovať práve jeden unikátny `SOA` záznam a aspoň jeden `NS` záznam. Kompletný záznam je štvorica: plne kvalifikované doménové meno, čas platnosti záznamu (*Time To Live* – skrátene *TTL*), trieda záznamu, typ záznamu a dáta záznamu. Trieda záznamu je väčšinou `IN`, čo znamená Internet (takmer nepoužívanými sú napríklad `CH`, *Chaos* alebo `HS`, *Hesoid*).

Prázdne miesta sú povolené v rámci celého súboru a sú brané ako oddelovače jednotlivých položiek. Komentáre sú uvedené znakom „;“ a končia znakom nového riadku. Direktíva `$ORIGIN` je nasledovaná doménovým menom, ktoré nahradí doposiaľ nastavenú hodnotu východzieho doménového mena. Východzia hodnota doménového mena je nastavená podľa názvu zónového súboru, v ktorom sa nachádza. Na východzie doménové meno sa odkazuje znakom „@“. Druhá direktíva `$INCLUDE` slúži na vloženie obsahu iného zónového súboru, pričom je možné vo voliteľnom parametre zadať, aké východzie doménové meno sa nastaví pre vkladaný obsah.

Popis niektorých typov záznamov:

- `A`, `AAAA`
  Prvý typ záznamu – `A` (*Address*) – slúži na preklad doménového mena na IPv4 adresu. Záznam typu `AAAA` slúži na preklad doménového mena na IPv6 adresu.
- `CNAME`

```
;komentar
$ORIGIN <domenove-meno> ;komentar
$INCLUDE <nazov-suboru> [<domenove-meno>]
<domenove-meno> [<TTL>] [<class>] <type> <RDATA>
                [<TTL>] [<class>] <type> <RDATA>

                ...

<domenove-meno> [<TTL>] [<class>] <type> <RDATA>
                [<TTL>] [<class>] <type> <RDATA>
```

Obr. 2.2: Formát zónového súboru

Canonical NAME záznam slúži ako alias pre dané meno. DNS systém následne hľadá odpoveď s uvedeným aliasom.

- `MX`
  Mail eXchange záznam mapuje doménové meno na zoznam mail serverov pre danú doménu.
- `NS`
  Name Server záznam definuje doménové mená autoritatívnych serverov pre danú doménu.
- `SOA`
  Záznam *Start Of Authority* musí byť uvedený v každom zónovom súbore, ktorý sa nachádza na rozhraní vlastnej a rodičovskej domény.

2.3.2 Konfigurácia serveru BIND

Server BIND (verzia 8 a vyššie) používa ku konfigurácii pri (re)štarte okrem zónových (popísaných vyššie) aj jeden konfiguračný súbor [4], ktorého názov je vo väčšine prípadov zachovaný: `/etc/named.conf`. Zónové súbory sa môžu nachádzať v ľubovolnom umiestnení, väčšinou je to však priečinok `/var/named`.

Konfiguračný súbor môže užívateľ vyplniť sám, alebo je možné použiť rôzne konfiguračné nástroje serveru BIND, ktoré tento súbor generujú. Server sa pri určitých chýbach v konfiguračnom súbore nespustí, niektoré chyby môžu byť odignorované. Kontrola syntaxe konfiguračného súboru je možná pomocou nástroja `named-checkconf`.

Formát konfiguračného súboru je vyobrazený na obrázku 2.3. Súbor je tvorený sekvenciami príkazov a komentárov. Každý príkaz je ukončený stredníkom.

Popis niektorých príkazov:

- `acl`
  Anglicky *access control list* povoľuje/zakazuje zoznam IP adries. Väčšinou sa používajú jednotlivé IP adresy alebo maska siete (napr. 10.0.1.0/20).
- `control`
  Definuje rôzne bezpečnostné požiadavky potrebné pre použitie nástroja `rndc`. Nástroj sa používa na vzdialenú administráciu DNS servera.

```
/*
  komentar-1
 */
// komentar-2
#  komentar-3
<prikaz-1> ["<prikaz-1-meno>"] [<prikaz-1-trieda>] {
   <prikaz-1> ["<prikaz-1-meno>"] [<prikaz-1-trieda>] {
      <volba-1>;
      <volba-2>;
      ...
      <volba-N>;
   };
   <volba-1>;
   <volba-2>;
   ...
   <volba-N>;
};

...

<prikaz-N> ["<prikaz-N-meno>"] [<prikaz-N-trieda>] {
   <prikaz-1> ["<prikaz-1-meno>"] [<prikaz-1-trieda>] {
      <volba-1>;
      <volba-2>;
      ...
      <volba-N>;
   };
   <volba-1>;
   <volba-2>;
   ...
   <volba-N>;
};
```

Obr. 2.3: Formát konfiguračného súboru servera BIND

- `include`
  Slúži na vkladanie obsahu iného súboru. Používa sa na príklad na vkladanie kľúčov, ktoré sú umiestnené v súboroch s inými užívateľskými právami.
- `key`
  Definuje určitý kľúč na základe jeho mena. Príkaz obsahuje voľbu šifrovacieho algoritmu a hodnotu kľúča.
- `trusted-keys`
  Obsahuje najrôznejšie verejné kľúče používané v DNSSEC (viac v kapitole 2.4).
- `server`
  Definuje určité vlastnosti, ktoré ovplyvňujú chovanie serveru voči vzdialeným DNS serverom, predovšetkým ohľadom notifikácií a prenosu zón.
- `view`
  Príkaz umožňuje nakonfigurovať server tak, aby na rovnaký dotaz odpovedal rôzne v závislosti od toho, kto sa dotazuje.
- `logging`
  Určuje logovacie kanály a ich úrovne, ktoré server zaznamená do logov o svojej činnosti.

- `options`
  Obsahuje najrôznejšie nastavenia servera ako pracovný adresár, mená určitých súborov a mnoho iného.

- `zone`
  Určuje zónu(y), pre ktorú je server autoritatívny. Príkaz označuje súbor obsahujúci dáta o danej zóne.

2.3.3 Konfigurácia serveru NSD

Implementácia autoritatívneho serveru NSD uplatňuje rovnaký prístup ako predošlé dva servery – okrem zónových súborov využíva jeden konfiguračný súbor, s názvom `nsd.conf` v priečinku `/etc/nsd`.

Formát konfiguračného súboru je zobrazený na obrázku 2.4. Povolené sú dve úrovne atribútov a k nim priradené hodnoty. Povolený je iba jeden typ komentárov uvedených znakom `#`.

```
1  # komentar
2  <atribut-1>:
3          <atribut-1>: <hodnota-1>
4          <atribut-2>: <hodnota-2>
5          ...
6          <atribut-N>: <hodnota-N>
7
8  ...
9
10 <atribut-N>:
11         <atribut-1>: <hodnota-1>
12         <atribut-2>: <hodnota-2>
13         ...
14         <atribut-N>: <hodnota-N>
```

Obr. 2.4: Formát konfiguračného súboru servera NSD

Atribútmi prvej úrovne môžu byť iba nasledujúce kľúčové slová (ktoré sa však môžu opakovať):

- `server`
  Sekcia obsahuje globálne nastavenia chodu serveru ako napríklad IP adresy, na ktorých server „počúva“, zoznam zónových súborov alebo nastavenia logovania.

- `key`
  Označuje kľúč s atribútmi ako meno, algoritmus a jeho hodnotou.

- `pattern`
  Na základe tejto „šablóny“ sú vybrané určité zóny, na ktoré sú dané vlastnosti aplikované. V tomto príkaze sa „šablóna“ pomenuje, pridajú sa jej vlastnosti a určí sa, na ktoré zóny sa „šablóna“ aplikuje.

- `zone`
  Príkaz je použitý pre každú zónu. Jeho atribútmi sú napríklad: meno zóny, umiestnenie zónového súboru, preferencie ohľadom notifikácií a podobne.

- `include`
  Používa sa na vloženie obsahu iného súboru.
- `remote-control`
  Definuje vlastnosti potrebné pre použitie vzdialeného ovládania serveru.

2.3.4 Konfigurácia serveru Knot DNS

Posledný z vybraných serverov je Knot DNS, ktorý sa drží konvencií a ku konfigurácii používa zónové súbory a jeden konfiguračný súbor, obvykle pomenovaný `knot.conf`. V blízkej budúcnosti by sa mal formát konfiguračného súboru zmeniť na jazyk *YAML*[5].

Formát konfiguračného súboru je zobrazený na obrázku 2.5. Pozostáva z vnorených blokov príkazov a ich hodnôt. Povolený je iba jeden typ komentárov uvedených znakom `#`.

```
# komentar
<prikaz-1> {
   <prikaz-1> {
      <volba-1> <hodnota-1>;
      <volba-2> <hodnota-2>;
      ...
      <volba-N> <hodnota-N>;
   }
   <volba-1> <hodnota-1>;
   <volba-2> <hodnota-2>;
   ...
   <volba-N> <hodnota-N>;
}

...

<prikaz-N> {
   <prikaz-1> {
      <volba-1> <hodnota-1>;
      <volba-2> <hodnota-2>;
      ...
      <volba-N> <hodnota-N>;
   }
   <volba-1> <hodnota-1>;
   <volba-2> <hodnota-2>;
   ...
   <volba-N> <hodnota-N>;
}
```

Obr. 2.5: Formát konfiguračného súboru servera Knot DNS

Príkazmi prvej úrovne môžu byť iba nasledujúce kľúčové slová:

- `system`
  Definuje všeobecné vlastnosti serveru ako napríklad verziu serveru, počet vlákien, maximálny počet paralelných spojení a podobne.
- `keys`
  Obsahuje TSIG kľúče použité pri autentizácii zónových prenosov.

[5]Skratka znamenajúca *YAML Ain't Markup Language*. Jedná sa o jazyk používaný na serializáciu dát, ktorý je pre čitateľa jednoduchší na porozumenie. `http://www.yaml.org/`

- `interfaces`
  Definuje IP adresy rozhraní, na ktorých Knot DNS „počúva“.
- `remotes`
  Nastavuje adresy vzdialených serverov/klientov pre zónové prenosy.
- `groups`
  Príkaz sa používa na vytvorenie skupín vzdialených serverov/klientov definovaných v sekcii `remotes`. Tieto skupiny môžu byť následne použité v konfigurácii, kde je možné uviesť vzdialený server/klienta.
- `control`
  Definuje IP adresy rozhraní, na ktorých server „počúva“ vzdialené príkazy.
- `zones`
  Príkaz definuje zóny, ktoré server obsluhuje. Jeho atribútmi sú napríklad meno zóny, umiestnenie zónového súboru, preferencie ohľadom notifikácií, vyžiadanie extra kontroly sémantiky zónového súboru a podobne.
- `log`
  Nastavuje vlastnosti logovania servera. Pre každý kanál (`stdout`, `stderr`, `syslog`) výstupu možno nastaviť úroveň detailov logovania, kategóriu (správy týkajúce sa behu servera, zón alebo všetkého) a umiestnenie logovacieho súboru.
- `include`
  Hodnota príkazu určuje súbor, ktorého obsah sa vloží na miesto príkazu.

2.4 Implementácia serveru Knot DNS

DNS server Knot DNS poskytuje viacero funkcií/vlastností. Viacero vlastností je spoločných spolu s inými implementáciami autoritatívnych DNS serverov. Všetky popísané vlastnosti je možné nastaviť v konfiguračnom súbore serveru. Popis hlavných funkcií/vlastností serveru Knot DNS [10]:

- **Inicializácia zónového prenosu**
  V systéme DNS existuje okrem komunikácie typu *resolver*[6]-server aj komunikácia medzi servermi. Táto komunikácia sa používa na proces prenosu zón medzi master a slave serverom. Server zahajuje prenos zónového súboru na základe dvoch prístupov. Pôvodným, dodnes používaným spôsobom, je takzvaný *polling*. Princípom pollingu je obnovovací interval. Každý zónový súbor obsahuje v zázname typu SOA hodnotu obnovovacieho intervalu, po vypršaní ktorého si slave server vyžiada od master servera aktuálnu verziu zónového súboru. Novším prístupom je technika notifikácií, popísaná v RFC 1996 [3]. Používa sa sériové číslo v SOA zázname. Pri zmene sériového čísla SOA záznamu v zónovom súbore rozpošle master server správu *NOTIFY* všetkým slave serverom pre danú zónu. Následne si slave servery vyžiadajú aktuálnu verziu zónového súboru. Server Knot DNS podporuje oba prístupy inicializácie.

[6] Program alebo súbor programov použitých na preklad doménových mien na IP adresy a opačne.

- **Zónový prenos *IXFR* a *AXFR***
  Po inicializácii zónového prenosu sa samotný zónový prenos rozdeľuje na dva typy. Prvý typ prenosu — *AXFR* (*asynchronous full transfer zone*) — sa týka prenosu celého zónového súboru, čo neni vhodné hlavne ak sa vo velkom zónovom súbore zmení minimálny počet záznamov. V tomto prípade je vhodnejší druhý typ zónového prenosu — *IXFR* (*incremental transfer zone*) — podporujúci prenos iba časti zónového súboru. V tomto prípade posiela slave server svoje aktuálne sériové číslo SOA záznamu. Master server na základe tohto sériového čísla vyhodnotí, ktoré časti je potreba zaslať. Prenesie sa teda iba časť zónového súboru. Server Knot DNS podporuje oba typy zónových prenosov.

- **Rezolúcia dotazu**
  Knot DNS server nepodporuje preposielanie DNS dotazov. Odpoveď na dotaz teda posiela priamo, nedotazuje sa iných DNS serverov. Pri vyhodnocovaní dotazu tento prejde viacerými fázami, ako napríklad predspracovanie dotazu, vyhodnotenie obsahu dotazu a hľadanie odpovedí a spracovanie odpovede pred odoslaním. Fázy a ich poradie určujú takzvané *query modules*.

- **Dynamické aktualizácie záznamov**
  V dnešnej dobe veľkú časť siete tvoria uzly s dynamicky prideľovanými IP adresami pomocou DHCP serverov. IP adresy týchto uzlov je potreba dynamicky viazať na konkrétne doménové mená v zónových súboroch. Mechanizmus dynamického aktualizovania DNS záznamov (angl. *Dynamic DNS updates*, *DDNS*) umožňuje automaticky tieto zmeny vykonávať. V systéme DNS na tento účel slúži správa *UPDATE* posielaná master serveru.

- **Zabezpečovacie mechanizmy**
  Systém DNS používa viacero zabezpečovacích mechanizmov. Mechanizmus DNSSEC (*DNS Security Extensions*) sa zaoberá zabezpečením integrity dát a mechanizmus TSIG (*Transaction SIGnature*) bezpečnosťou prenosu záznamov pri prenose zón alebo pri dynamickej aktualizácii DNS záznamov. Zatiaľ čo TSIG zabezpečenie používa symetrickú kryptografiu, DNSSEC využíva asymetrickú. Zatiaľ čo TSIG nepotrebuje k svojej činnosti žiadne nové záznamy v zónových súboroch, DNSSEC ich využíva hneď niekoľko [6]. Popis najpoužívanejších záznamov:

  - *DNSKEY*: využíva sa pri podpisovaní a autentizácii ostatných DNS záznamov
  - *RRSIG*: obsahuje elektronický podpis množiny záznamov, ktoré majú rovnaké doménové meno, typ, triedu a TTL
  - *DS*: uložený v rodičovskej zóne, slúži na overenie pravosti DNSKEY kľúča
  - *NSEC*: vytvára reťazec doménových mien, čím vlastne autoritatívne potvrdzuje neexistenciu niektorých záznamov v doméne
  - *NSEC3*: vylepšená verzia pôvodného mechanizmu NSEC, ktorá bráni načítaniu všetkých doménových mien zóny
  - *NSEC3PARAM*: obsahuje parametre potrebné pre vypočítanie hashovacej funkcie

- **NSID**
  Anglicky *Name Server IDentifier* je jednoznačný identifikátor konkrétneho DNS ser-

veru z množiny serverov s rovnakou IP adresou. Rovnaká IP adresa sa používa kvôli geografickému rozloženiu DNS serverov, kedy môžu byť dotazy smerované najbližším serverom (zabezpečujú smerovacie protokoly na sieťovej vrstve).

- **Response Rate Limiting**
  Ochrana voči útoku, kedy prijímaný UDP paket neobsahuje adresu odosielateľa. Tento parameter obmedzuje počet odpovedí servera za sekundu.

Zhrnutie kapitoly

Kapitola postupne v jednoduchosti a bez väčších detailov vysvetlila čo je to autoritatívny DNS server, ktoré implementácie DNS serverov táto práca popisuje, aké sú možnosti ich konfigurácie a popis základných používaných mechanizmov a prebiehajúcich procesov v implementáciách autoritatívnych DNS serverov. V časti konfigurácii serverov bol popísaný formát a význam zónových súborov a konfiguračných súborov, ktoré jednotlivé DNS servery používajú.

Kapitola 3

Spôsoby testovania DNS serverov

Testovanie tvorí dôležitú časť životného cyklu softwaru, ktorá je však často krát podceňovaná. Prečo je testovanie také dôležité? Dáva nám totižto odpoveď na dve dôležité otázky: „Ako kvalitný software je a dosiahol už úrovne kedy ho zákazník príjme?“ a „Kde software obsahuje chyby?“. Tak isto ako každý iný druh softwaru aj implementácie DNS serverov je potrebné dôkladne testovať.

Kapitola uvádza základy funkcionálneho, štrukturálneho a integračného testovania. Obsah týchto kapitol je voľne prevzatý z kníh autorov Paula C. Jorgensena - *Software testing* [5] a Rona Pattona - Testovanie SW [7]. Koniec kapitoly je venovaný teórii o statickom a dynamickom testovaní.

3.1 Funkcionálne testovanie

Funkcionálne testovanie je založené na prístupe, kedy každý program môže byť považovaný za funkciu, ktorá mapuje hodnoty z vstupu na hodnoty výstupu. Tento prístup je používaný prevažne v technickom prostredí a nazýva sa tiež *black box* testovanie. Hlavným rysom tohto typu testovania je fakt, že človek, ktorý testy vytvára, vidí systém ako „čiernu skrinku“. Do systému teda iba predáva vstupy a sleduje výstupy. Na základe porovnania výstupov s referenčnými sa zisťuje, kde v systéme sa vyskytuje chyba.

Výhodou funkcionálneho testovania je vlastnosť, že je nezávislý od implementácie. Testy vytvorené pre jednu implementáciu môžu byť teda neskôr využité aj pre inú implementáciu. Taktiež vývoj testov môže prebiehať paralelne s vývojom softwaru, čo ušetrí celkový čas jeho vývoja. Na druhej strane trpí funkcionálne testovanie dvoma problémami: medzi testami môžu existovať značné redundancie spolu s medzerami neotestovaných častí softwaru.

3.2 Štrukturálne testovanie

Štrukturálne testovanie je niekedy taktiež nazývané *white box* testovanie. Tento typ sa používa na verifikovanie zdrojového kódu. Berie do úvahy obsah „bielej skrinky“ a je teda možné lepšie nájsť také dáta, ktoré povedú k chybe v softwari. Nevýhodou je možné riziko,

že človek, ktorý píše testy môže tieto prispôsobiť tak, aby na danej implementácii vyhoveli. Tomuto riziku sa snaží vyhnúť druhý typ testovania, integračné testovanie, popísaný v nasledujúcej podkapitole.

3.3 Integračné testovanie

Tento druh testovania slúži na testovanie spojených prvkov systému. V najjednoduchšej forme je to testovanie komponenty systému agregovanej z dvoch prvkov systému, ktoré už boli samostatne otestované. Zložené komponenty teda môžu byť zložené z dvoch alebo viacerých agregovaných prvkov systému. V praxi sú prvky spájané do malých komponent a tie následne do väčších, ktoré sú testované. Zámerom je otestovanie rôznych kombinácií komponent a následne celých modulov.

Integračné testovanie identifikuje chyby, ktoré sa vyskytujú pri spájaní komponent. Použitím testovacieho plánu, ktorý vyžaduje, aby bol každý prvok systému dôkladne otestovaný pred tým, ako je agregovaný, dosiahneme toho, že chyby, ktoré nastanú pri spájaní, sú veľmi pravdepodobne spojené s nekompatibilitou rozhraní agregovaných prvkov.

Integračné testovanie možno vykonávať rôznymi spôsobmi. Medzi najčastejšie stratégie patria:

- **Zhora-dolu**:
  Prístup zhora-dolu vyžaduje ako prvé vykonanie integračného testovania na hierarchicky najvyšších moduloch. To umožňuje otestovanie logiky na najvyššej úrovni spolu spolu tokom dát v rannom štádiu vývoja softwaru. Potreba náhrady logiky v testovaných komponentách pomocou *stubs*[1] programov komplikuje testovanie a prvky na najnižšej úrovni sú testované relatívne neskoro vzhľadom na vývojový cyklus softwaru. Vďaka tomu je vydávanie skorých funkčných verzií softwaru veľmi obmedzené.
- **Zdola-hore**:
  V kontraste s prvým prístupom, prístup zdola-hore vyžaduje integračné testovanie najzákladnejších prvkov systému. Základné prvky systému sú teda vyvíjané ako prvé, a minimalizuje sa tým potreba stubs programov. Logika na najvyššej úrovni spolu s tokom dát je však testovaná až v poslednej fáze životného cyklu. Podobne ako u predošlého prístupu testovania je skoré vydávanie funkčných verzií softwaru veľmi obmedzené.
- **Kombinovaný**:
  Prístup, anglicky tiež nazývaný ako *sandwich*, kombinuje oba predošlé prípady. Preto je vhodný na vydávanie skorých verzií softwaru.

3.4 Statické a dynamické testovanie

K testovaniu softwarového produktu môžeme pristupovať viacerými spôsobmi. Na základe toho, či je potreba software v priebehu testovania spúšťať, definujeme dva typy testov:

[1] *Stub* je program, ktorý simuluje chovanie komponenty/prvku systému.

- **Statické testy**:
  Nevyžadujú k svojmu vykonaniu spustiteľný software, preto je možné ich vykonávať ešte pred samotným funkčným prototypom softwaru. Statické testy predstavujú proces verifikácie.
- **Dynamické testy**:
  K svojmu vykononaniu potrebujú funkčný software schopný behu. Testy totižto prebiehajú nad funkčnou implementáciou, ktorej poskytujú predom určené vstupy a kontrolujú jej výstupy. Dynamické testy predstavujú proces validácie.

Zhrnutie kapitoly

Framework, o ktorom pojednáva táto písomná práca, je navrhnutý pre dynamické *black box* testovanie a DNS servery sú chápané ako „čierna skrinka“. Tento spôsob testovania je výhodný vďaka svojej neviazanosti na konkrétnu implementáciu DNS servera a možno ho teda použiť na testovanie iných DNS serverov.

Z pohľadu testov sa vo frameworku používajú dynamické testy. Keďže sa testuje beh DNS serveru, je k testovaniu potrebný spustiteľný software. Beh DNS serveru je ovplivňovaný iba vstupmi, teda DNS dotazmi, prenosmi zón, ich konfiguráciami a podobne, ktoré upravujú jednotlivé testovacie scenáre.

Kapitola 4

Návrh testovacieho frameworku

Náplňou tejto práce je úprava stávajúceho frameworku vyvíjaného združením CZ.NIC. Framework je vyvíjaný spolu so serverom Knot DNS a jeho primárnym cieľom je testovanie implementácie tohto servera. Framework je distribuovaný spolu so serverom Knot DNS. Získať ho možno stiahnutím celého archívu na webových stránkach CZ.NIC alebo z Github repozitáru[1]. Testovací framework sa nachádza v priečinku `/tests-extra`.

Stav frameworku pred úpravou nebol vyhovujúci z viacerých dôvodov (kap. 4.4) a preto vznikla požiadavka na jeho úpravu. Spoločným cieľom úprav bolo zjednodušenie a spríjemnenie používania.

V nasledujúcich podkapitolách bude popísaný stávajúci framework serveru Knot DNS (kap. 4.1), jeho štruktúra (kap. 4.2), spôsob používania frameworku (kap. 4.3) a na koniec budú uvedené požiadavky na upravený framework spolu s ich náčrtom riešenia (kap. 4.4).

4.1 Stav frameworku

Prevažná väčšina stávajúceho frameworku je implementovaná v skriptovacom jazyku Python 3 a pomocné skripty sú implementované v jazyku Bash. Táto kombinácia bude použitá aj v upravenom frameworku vďaka výhodám v oblasti dynamického typovania, ktoré poskytuje jazyk Python 3.

Framework používa objektovo orientovanú knižnicu jazyka Python 3 – `dnspython`[2]. Knižnica obsahuje sadu nástrojov pre prácu s DNS systémom. Podporuje dotazy, zónové transfery, väčšinu typov DNS záznamov a iné. Ponúka ako nízko tak aj vysokoúrovňový prístup k systému DNS. Zatiaľ čo na nízkej úrovni je možné priamo manipulovať so správami, zónami alebo DNS záznamami, vysoká úroveň prístupu poskytuje dostatočný stupeň abstrakcie na jednoduché vytvorenie DNS dotazu a získanie jeho odpovede.

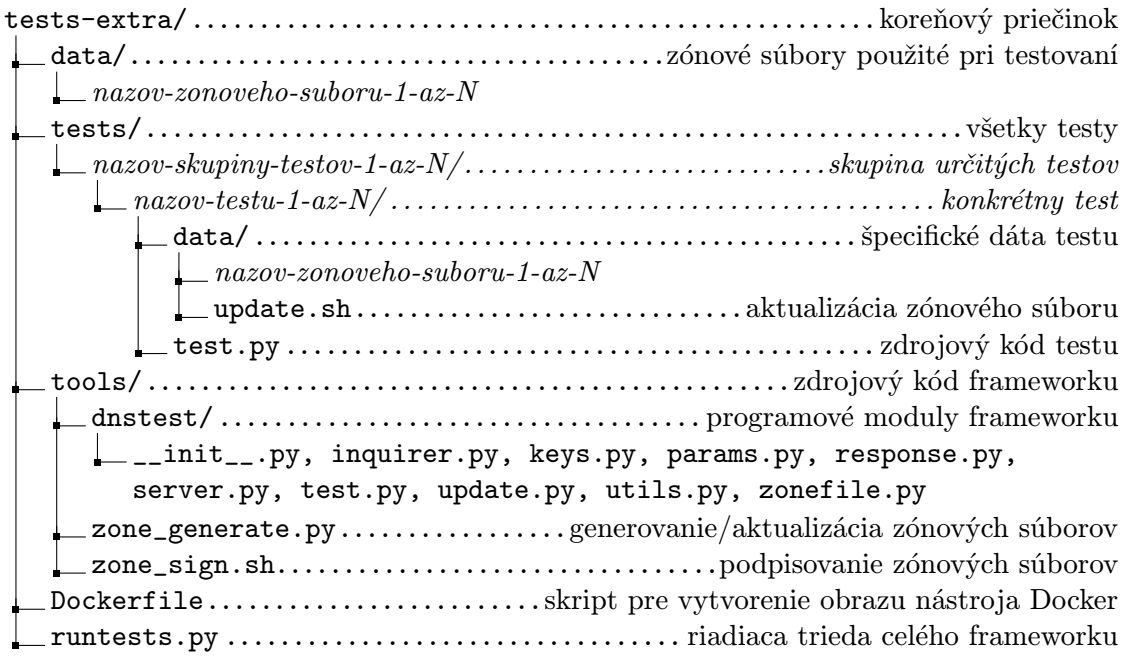
Východzí framework v základe podporuje testovanie implementácií DNS serverov Bind a Knot DNS.

[1] `https://gitlab.labs.nic.cz/labs/knot/`
[2] `http://www.dnspython.org/`

Keďže framework podlieha neustálemu vývoju zo strany asociácie CZ.NIC, stav pôvodného frameworku popisovaný v tejto a nasledujúcich podkapitolách sa nemusí zhodovať s aktuálnym stavom frameworku dostupného na Github repozitári. Popisovaný framework je aktuálny k dátumu 1.4.2015, teda v čase písania tejto práce.

4.2 Štruktúra frameworku

Koreňový adresár frameworku vzhľadom ku koreňovému adresáru implementácie serveru Knot DNS sa nachádza v adresáre `/tests-extra`. Umiestnenie všetkých nasledujúcich súborov v tejto kapitole uvedených pomocou relatívnej cesty sa bude odkazovať na tento koreňový priečinok. Štruktúra koreňového adresáru spolu s heslovitým popisom je zobrazená na obrázku 4.1. Jednotlivé programové moduly budú popísané detailnejšie v kapitole 4.2.1.



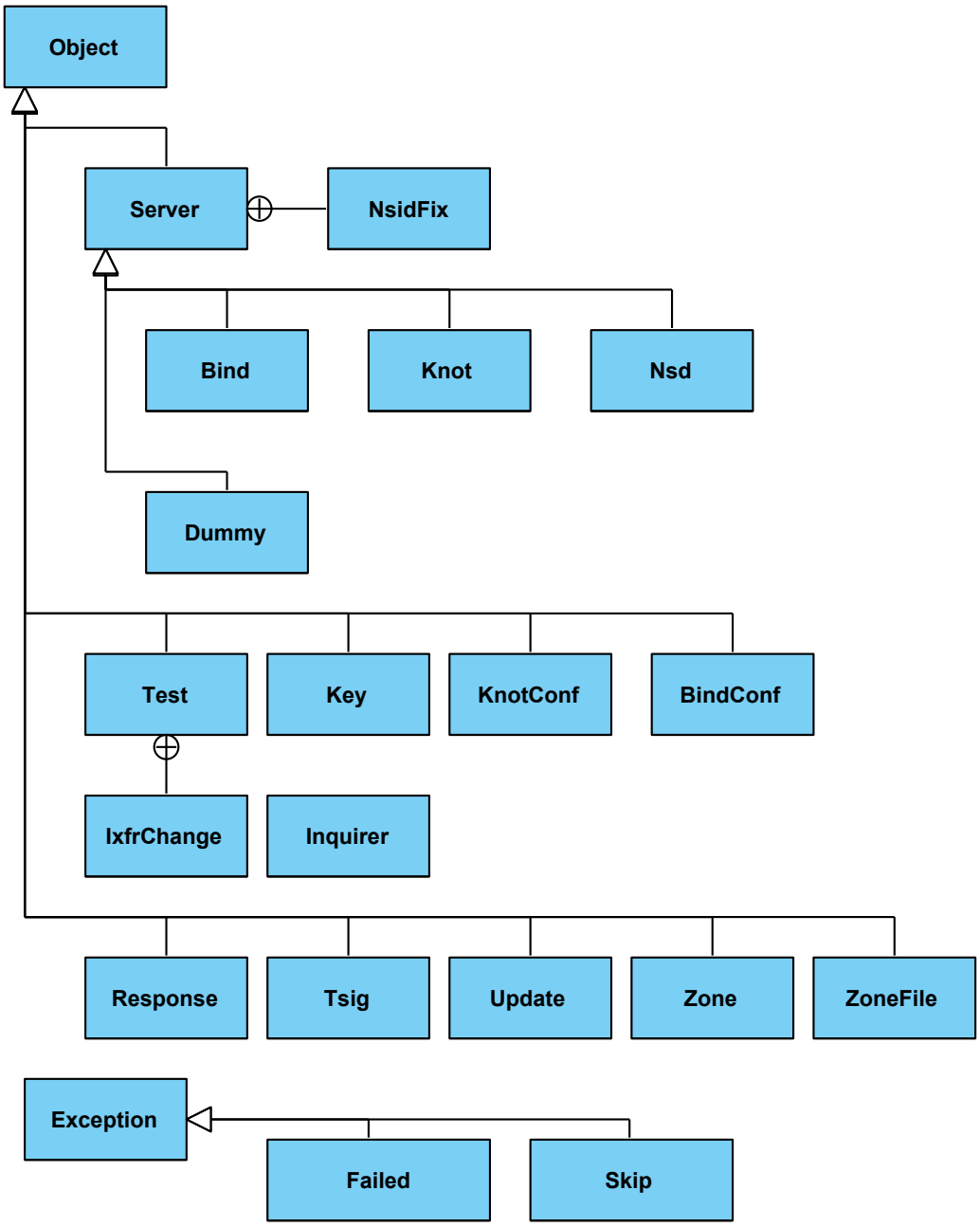
Obr. 4.1: Štruktúra koreňového adresára frameworku

Stručný popis významných prvkov štruktúry celého frameworku:

- `./data`
  Obsahuje viacero rôznych zónových súborov, ktoré sa používajú v rôznych testoch na testovanie špecifických vlastností serveru.
- `./tests`
  Priečinok obsahuje množiny testov, rozdelené do skupín. Každý test je implementovaný v súbore `test.py`, ktorý daný test vykonáva.
- `./tools`
  Obsahuje zdrojový kód frameworku a jeho moduly.

4.2.1 Moduly

Stávajúci framework je rozčlenený do modulov. Niektoré z modulov obsahujú viacero tried. Tieto triedy sú vyobrazené na nasledujúcom diagrame tried 4.2.



Obr. 4.2: Diagram tried stávajúceho frameworku

Diagram, respektíve framework, obsahuje dve Python triedy. Trieda `object`[3] slúži ako bázová trieda pre všetky vytvárané triedy v Pythone. Obsahuje metódy spoločné pre všetky inštancie triedy v Pythone. Druhou je bázová trieda pre programové výnimky `Exception`[4]

[3] `https://docs.python.org/3.3/library/functions.html#object`
[4] `https://docs.python.org/3.3/library/exceptions.html#Exception`

potrebná pre vytvorenie vlastných tried `Failed` a `Skip` reprezentujúcich výnimky. Popis ostatných programových modulov stávajúceho frameworku spolu s popisom obsahujúcich tried:

- `./runtests.py`
  Ide o hlavný skript, ktorý spúšťa všetky zadané testy a riadi ich priebeh. Zoznam názvov testov je predaný pomocou parametrov príkazového riadku.
- `./Dockerfile`
  Skript slúži na automatické vytvorenie obrazu pre nástroj Docker[5].
- `./tools/zone_generate.py`
  Skript vytvorí alebo upraví existujúcu zónu. Zónový súbor je možné poprípade podpísať. V tom prípade sa použije skript `./tools/zone_sign.sh`.
- `./tools/zone_sign.sh`
  Bash skript, ktorý podpisuje zóny. Za týmto účelom používa nástroje `dnssec-keygen` a `dnssec-signzone`.
- `./tools/dnstest/__init__.py`
  Pythonovský index pre balíček `dnstest`.
- `./tools/dnstest/inquirer.py`
  Trieda `Inquirer` vytvára procesy, ktoré sa následne dotazujú DNS serverov dané alebo náhodné DNS dotazy. Cieľom je umelé zvýšenie vyťaženia servera.
- `./tools/dnstest/keys.py`
  Modul obsahuje tri triedy: `Tsig` a `Key`. Trieda `Tsig` slúži na generovanie TSIG kľúčov používaných pri podpisovaní komuníkácie medzi DNS servermi. Druhá v poradí, trieda `Key`, vytvára DNSSEC kľúče.
- `./tools/dnstest/params.py`
  Skript definuje globálne premenné zdieľané medzi modulmi. Premenné obsahujú informácie ako umiestnenie binárnych súborov nástroja `valgrind`, serverov BIND, Knot DNS a podobne.
- `./tools/dnstest/response.py`
  Trieda `Response` ukladá prijaté odpovede od DNS serverov. Obsahuje funkcie na kontrolu odpovedí voči referenčným odpovediam – tieto sú určené užívateľom (najčastejšie sa porovnáva voči odpovediam od serveru BIND).
- `./tools/dnstest/server.py`
  Skript obsahuje viacero tried. Trieda `Zone` reprezentuje abstrakciu zóny. Bázová trieda `Server` definuje spoločné rysy implementácií DNS serverov, funkcie pre štart, stop alebo reload servera, funkcie pre vytváranie dotazov a ich odosielanie a podobne. Z tejto bázovej triedy dedia triedy `Knot`, `Bind`, `Nsd` a `Dummy`. Modul obsahuje taktiež triedy `KnotConf` a `BindConf` reprezentujúce konkrétne konfigurácie DNS serverov BIND a Knot DNS. Slúžia na generovanie konfiguračných súborov pre dané implementácie DNS serverov. Posledná je trieda `NsidFix` vytvorená pre potreby knižnice `dnspython`.

[5] `https://www.docker.com/`

- `./tools/dnstest/test.py`
  Modul zastrešuje rozhranie frameworku pomocou triedy `Test`. Pomocou neho sa získavajú objekty reprezentujúce servery, vykonáva sa správa serverov použitých na testovanie (štart, stop, počiatočná konfigurácia, ukončenie, ...), generovanie a priraďovanie zón (master-slave vzťah) a podobne. Trieda `IxfrChange` reprezentuje zmeny vykonané v zóne pomocou inkrementálneho transferu zón. Obsahuje zoznam pridaných a odstránených záznamov.
- `./tools/dnstest/update.py`
  Trieda `Update` tohto modulu implementuje „wrapper“ pre triedu `dns.update.Update`.
- `./tools/dnstest/utils.py`
  Implementuje rôzne pomocné funkcie a dve triedy predstavujúce výnimky – `Failed` a `Skip`.
- `./tools/dnstest/zonefile.py`
  Trieda `ZoneFile` predstavuje jeden zónový súbor v súborovom systéme.

4.2.2 Testy

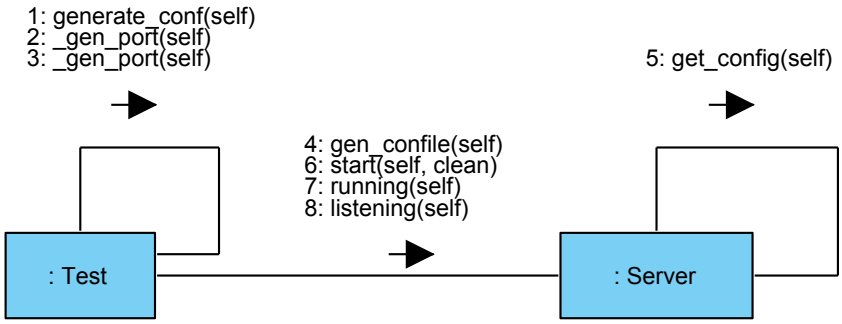
Framework obsahuje viacero testov zameraných na testovanie špecifických vlastností serveru Knot DNS. Testy vznikajú za účelom testovania nových vlastností, ktoré sa pridávajú do implementácie novej verzie servera. Medzi existujúce testy patria napríklad testy na prenos zón (AXFR, IXFR), ich zabezpečenie (NSEC, NSEC3), rôzne typy dotazov, testovanie systému *Dynamic DNS*, EDNS, testovanie práce so zónovými súbormi a iné.

Testy sú rovnako implementované v jazyku Python 3. Každý test je reprezentovaný samostatným skriptom, ktorý implementuje chovanie testu. Každý testovací skript sa približne drží nasledujúcej schémy implementácie:

1. vytvorenie objektu triedy Test
2. získanie objektov reprezentujúcich jednotlivé implementácie DNS serverov z vytvoreného objektu tredy Test
3. konfigurácia serverov
4. (vygenerovanie a) nastavenie zónových súborov serverov
5. spustenie testu pomocou objektu triedy Test
6. konkrétne príkazy špecifické pre testovanie jednotlivých vlastností serverov
7. ukončenie testu pomocou objektu triedy Test

Spúšťanie testu zaobstaráva trieda `Test`, konkrétne funkcia `start`. Diagram komunikácie tejto funkcie je na obrázku 4.3.

Framework pri zahájení behu testu postupne spúšťa jednotlivé implementácie použitých DNS serverov. O tento proces spúšťania sa stará trieda `Test`, pričom postupuje nasledovným spôsobom:



Obr. 4.3: Diagram komunikácie spúšťania testu

1. Vygenerovanie obsahu konfiguračného súboru a jeho zapísanie do súboru na určené miesto. Tento krok je vykonaný pre každý server, ktorý je použitý v danom teste.
2. Zoradenie testovaných DNS serverov na základe ich priority nasledovne (zoradené podľa priority zostupne): servery slúžiace ako master servery pre zóny a servery slúžiace ako slave servery. Poradie v rámci master serverov je určené počtom zón, pre ktoré je server master serverom. Teda čím viac zón, tým vyššia priorita. Poradie v rámci slave serverov je zachované na základe poradia ich vytvorenia cez rozhranie triedy `Test`.
3. Spustenie implementácie DNS servera.
4. Kontrola behu procesu DNS servera pomocou Python balíčka `psutil`[6].
5. Kontrola naviazania DNS servera na zadané porty. Podľa identifikačného čísla procesu bežiaceho DNS serveru sa kontrolujú otvorené sokety. Vo väčšine prípadov sa kontrolujú sokety pre TCP a UDP protokol. Funkcia na určenie otvorených soketov je implementovaná v triedach reprezentujúcich DNS servery.
6. V prípade, že sa DNS server nepodarilo spustiť, proces spúšťania sa opakuje do zadaného počtu krát. Po vyčerpaní pokusov pre spustenie je test ukončený chybou.

4.3 Použitie stávajúceho frameworku

Stávajúci framework slúži na testovanie hlavne serveru Knot DNS. Framework testuje implementáciu servera pravidelne každú noc, kedy je testovací server najmenej vyťažený. Následne je po skončení testovania potrebná manuálna kontrola súhrnného logu, ktorý označuje úspešné a neúspešné vykonanie testov. Umiestnenie všetkých nasledujúcich súborov v tejto kapitole uvedených pomocou relatívnej cesty sa bude odkazovať na priečinok `./tests-extra`.

Framework požaduje nainštalované prerekvizity uvedené v súbore `./README` a prítomnosť nainštalovaných serverov DNS, ktoré budú predmetom testovania. Keďže je potreba výstupy testovaného serveru nejakým spôsobom overiť, je minimálny rozumný počet DNS serverov

[6]Skratka `psutil` (anglicky *python system and process utilities*). Balíček obsahuje prostriedky (funkcie, objekty, a pod.) na zistenie informácií o bežiacich procesoch a vyťažení výpočetných prostriedkov hostiteľského systému. Viac na `https://pythonhosted.org/psutil/`.

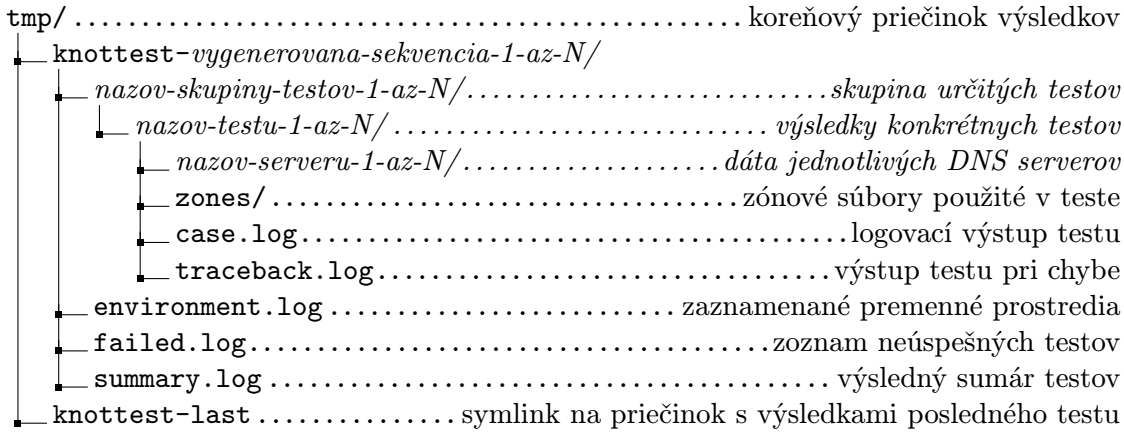
dva (keďže porovnávať jednu a tú istú implementáciu DNS serveru s tou istou by neposkytlo žiadne výsledky). Stávajúce testy používajú vo väčšine prípadov ako referenčný DNS server implementáciu BIND, testovaným serverom je Knot DNS.

Pre spustenie testovania slúži skript `./runtests.py`. Skript berie ako parametre zoznam testov, ktoré majú byť vykonané, poprípade zoznam testov, ktoré majú byť vynechané. Na základe predaných parametrov začne spúšťať jednotlivé testovacie skripty a zaznamenáva ich priebeh vykonávania. Výstupom behu frameworku sa zaoberá nasledujúca kapitola 4.3.1.

4.3.1 Výstupy testov

Každé spustenie frameworku vytvorí vo vybranom priečinku adresárovú štruktúru, do ktorej sú následne uložené výsledky priebehu testov. Umiestnenie tohto priečinku je možné zmeniť priamo v skripte `./tools/dnstest/params.py` v premennej `outs_dir` alebo nastavením premennej prostredia `KNOT_TEST_OUTS_DIR` na požadované umiestnenie.

Adresárová štruktúra na obrázku 4.4 bola vytvorená s prednastaveným umiestnením v priečinku `/tmp/`. Priečinok `knottest-`*vygenerovana-sekvencia-1-az-N* reprezentuje jeden beh frameworku. V priečinku sa nachádzajú podpriečinky obsahujúce jednotlivé množiny testov. Na ďalšej úrovni sú to už výsledky konkrétnych testov a iné dáta použité pri testoch (konfiguračné súbory serverov, zónové súbory, chybový/štandardný výstup serverov, kľúče a iné). Súbor `case.log` obsahuje logovací výstup konkrétneho testu a `traceback.log` obsahuje chybový výstup interpreta jazyka Python 3.



Obr. 4.4: Štruktúra adresára výsledkov testov

Priečinok `knottest-`*vygenerovana-sekvencia-1-az-N* obsahuje ešte tri logovacie súbory. Do súboru `environment.log` sa zaznamenávajú hodnoty premenných prostredia použitých pri behu, súbor `failed.log` obsahuje zoznam neúspešných testov s jednoduchým popisom chyby a posledný súbor `summary.log` obsahuje jednoduchý zápis behu frameworku spolu s časovými razítkami jednotlivých udalostí.

V prípade testov, kedy je server dotazovaný na záznamy priamo z rozhrania frameworku (metóda `Server.dig()`), sú jednotlivé dotazy zaznamenávané do súboru `case.log`. Tento súbor teda v prípade chyby obsahuje záznam, pri ktorom dotaze k chybe došlo.

Symbolický odkaz `knottest-last` slúži na prístup do priečinku obsahujúceho výsledky posledného vykonaného behu frameworku.

4.4 Požiadavky na upravený framework

Hľavným cieľom tejto práce je úprava stávajúceho testovacieho frameworku, ktorý je primárne určený na testovanie serveru Knot DNS. Framework slúži na overenie správnosti zmien v implementácii. Avšak vďaka viacerým obmedzeniam bol framework kandidátom na viaceré zmeny. Tie sa týkali jednak výkonnostnej stránky (paralelizácia) ale aj stránky funkčnej (podpora viacerých impl. serverov, rozdelenie do viacerých modulov, ...). Hlavné požadované zmeny a ich navrhované riešenia sú popísané v nasledujúcich podkapitolách.

4.4.1 Rozdelenie na komponenty

Stávajúci framework obsahuje viacero zdrojových súborov obsahujúcich viacero tried, ktoré by mohli byť rozdelené do samostatných súborov kvôli prehľadnosti. Najvhodnejším kandidátom je súbor `server.py`, ktorý obsahuje triedy reprezentujúce jednotlivé implementácie serverov, abstraknú triedu servera, triedy reprezentujúce konfigurácie jednotlivých serverov a triedu obsahujúcu popis zóny.

4.4.2 Flexibilita pri zápise testov

Umožniť väčšiu flexibilitu pri zápise testov. Teda umožniť vykonanie špeciálnej operácie, ktorá neni priamo implementovaná frameworkom. Napríklad špecifické úpravy konfigurácie serveru bude možné vykonávať cez abstraknú triedu konfigurácie serveru.

4.4.3 Podpora ďalších implementácií DNS serverov

Stávajúci framework podporuje implementácie BIND a Knot DNS. Cieľom je úprava frameworku do stavu, kedy bude možné jednoducho pridávať ďalšie implementácie vo forme modulov/tried, ktoré budú implementovať určité rozhranie a dediť z určitých abstraktných tried (základná trieda servera a konfigurácie). Výsledný framework by mal podporovať minimálne tieto tri implementácie DNS serverov: BIND, Knot DNS a NSD.

4.4.4 Paralelizácia testov

Pôvodný framework implicitne nepodporuje možnosť paralelizácie testov, podporuje iba sériové vykonávanie zadaných testov. Samotnú paralelizáciu je potrebné riešiť v spojení s testami. Paralelizáciu by bolo možné vykonať na 3 úrovniach:

1. Na úrovni frameworku – celý framework je ako celok spustený viackrát paralelne, každá inštancia dostane na testovanie iba podmnožinu testov.

2. Na úrovni testov – framework ako celok je spustený iba jeden krát a zadané testy sú vykonávané paralelne. V tomto prípade je však ďalšou prekážkou fakt, že paralelné vykonávanie jednotlivých testov by mohlo ovplyvniť výsledky ostatných testov, v prípade, že by bol použitý jeden DNS server.
3. Na úrovni testovacích príkazov, resp. sekcie testovacích príkazov – v jednotlivých testoch by boli označené príkazy alebo sekcie príkazov, ktoré by bolo možné vykonať paralelne. Tieto sekcie by boli označené užívateľom priamo v testoch.

Vhodným riešením by mohla byť paralelizácia na úrovni frameworku, pričom by bola použitá technika takzvaných „kontajnerov“.

4.4.5 Vytvorenie *dummy* serveru

V niektorých prípadoch testovania je potreba posielať predpripravené odpovede testovaným serverom. Pôvodný framework túto možnosť podporuje iba nízkoúrovňovým prístupom ku knižnici `dnspython` a manuálnym vytváraním odpovedí. Vytvorením triedy *dummy* (v preklade model alebo replika) serveru s vlastnou čiastočnou inteligenciou a jednoduchým nakonfigurovaním chovania by bolo možné dosiahnuť požadovaného efektu.

4.4.6 *Blackbox*

Implementácia chýbajúceho prvku *Blackbox*, cez ktorý by mala byť vedená komunikácia medzi dvoma DNS servermi. Užívateľ by mal mať možnosť pristupovať k týmto dátam a ľubovolne s nimi manipulovať. To znamená, že dáta môžu byť ľubovolne upravené, vymazané, oneskorené a podobne. *Blackbox* by teda mohol slúžiť na simulovanie chýb pri prenose ako napríklad vytváranie chýb na úrovni bitov alebo DNS správ, vytváranie latencie linky medzi servermi a podobne.

4.4.7 Vytvorenie exemplárnych testov

Nad upraveným frameworkom je potreba vytvoriť základnú množinu testov pre demonštráciu požadovaných funkcií a vykonaných zmien. Ide o základné testy pre prenos zón, dotazy a podobne.

Kapitola 5

Úpravy frameworku pre testovanie DNS serverov

Pôvodný projekt frameworku umiestnený na úložisku Github[1] bol naklonovaný do lokálneho úložiska. Na účely úpravy/vývoja frameworku bola vytvorená vetva `test_framework_remake`, ktorá je uložená na priloženom CD (viď. kap. A).

Nasledujúce podkapitoly popisujú zmeny, ktoré boli vykonané na pôvodnom frameworku. Každá kapitola opisuje sledovaný cieľ úpravy, komponenty frameworku, ktoré boli upravené, spôsob akým boli upravené a akých výsledkov bolo dosiahnutých vykonaním úprav.

5.1 Rozdelenie na komponenty

Zadaná požiadavka

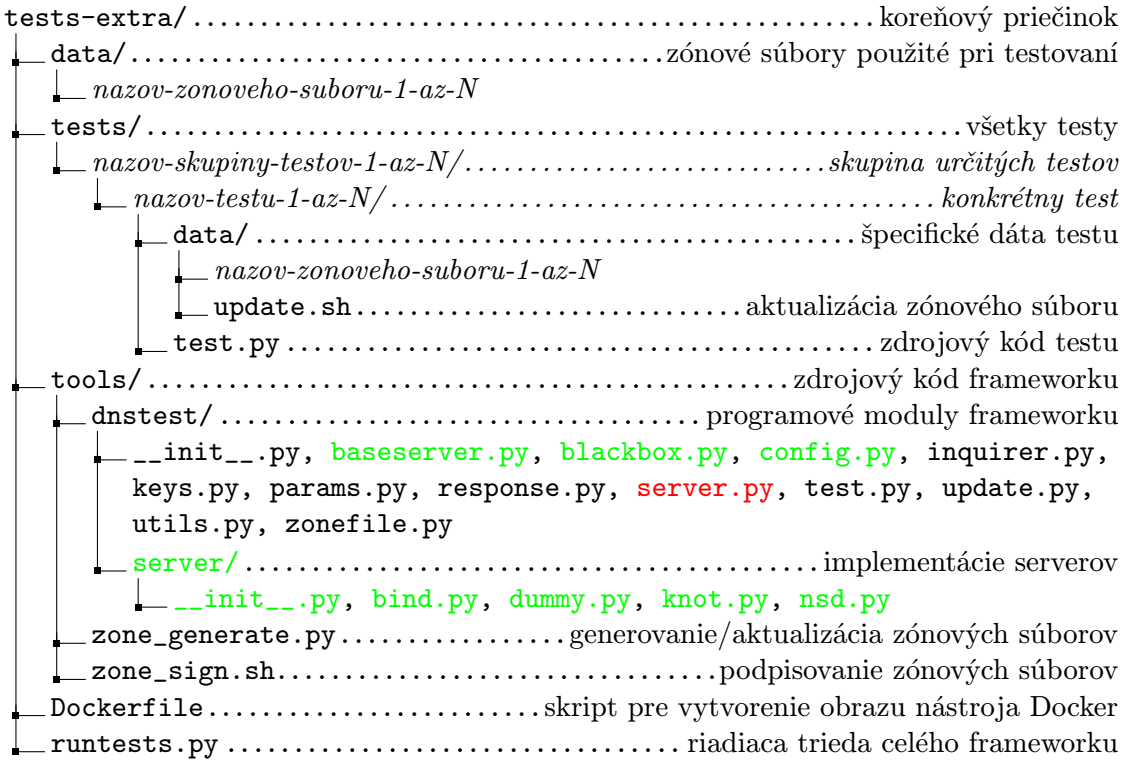
Stávajúci framework obsahuje viacero zdrojových súborov obsahujúcich viacero tried, ktoré by mohli byť rozdelené do samostatných súborov kvôli prehľadnosti a lepšej hierarchii.

Riešenie požiadavky

Najvhodnejším kandidátom je súbor `server.py`, ktorý obsahuje triedy reprezentujúce jednotlivé implementácie serverov, abstraknú triedu servera, triedy reprezentujúce konfigurácie jednotlivých serverov a triedu obsahujúcu popis zóny. Prvou vykonanou zmenou teda bolo rozdelenie modulu `server.py` na moduly, ktoré reprezentujú jednotlivé servery (Bind, Knot DNS, ...). Súbory obsahujúce jednotlivé moduly serverov boli presunuté do priečinku `server`. Z modulov bol následne vytvorený Python-ovský balíček `dnstest.server.*`. Toto rozčlenenie slúži ako predpríprava na úpravu podporujúcu viacero implementácií DNS serverov (kap. 5.3).

Pôvodná štruktúra frameworku popísaná v kapitole 4.2 sa zmenila na štruktúru popísanú na obrázku 5.1. Prvky označené červenou farbou boli zrušené alebo presunuté na iné miesto a prvky označené zelenou farbou boli vytvorené alebo presunuté na konkrétne miesto.

[1] `https://gitlab.labs.nic.cz/labs/knot/`



Obr. 5.1: Štruktúra koreňového adresára upraveného frameworku

Výsledky a zhrnutie

Výsledkom dekompozície modulov je dosiahnutie lepšieho hierarchického členenia frameworku. Výsledný diagram tried frameworku je na obrázku 5.2. Triedy vyznačené zelenou farbou sú tie, ktoré boli oproti stávajúcemu frameworku vytvorené.

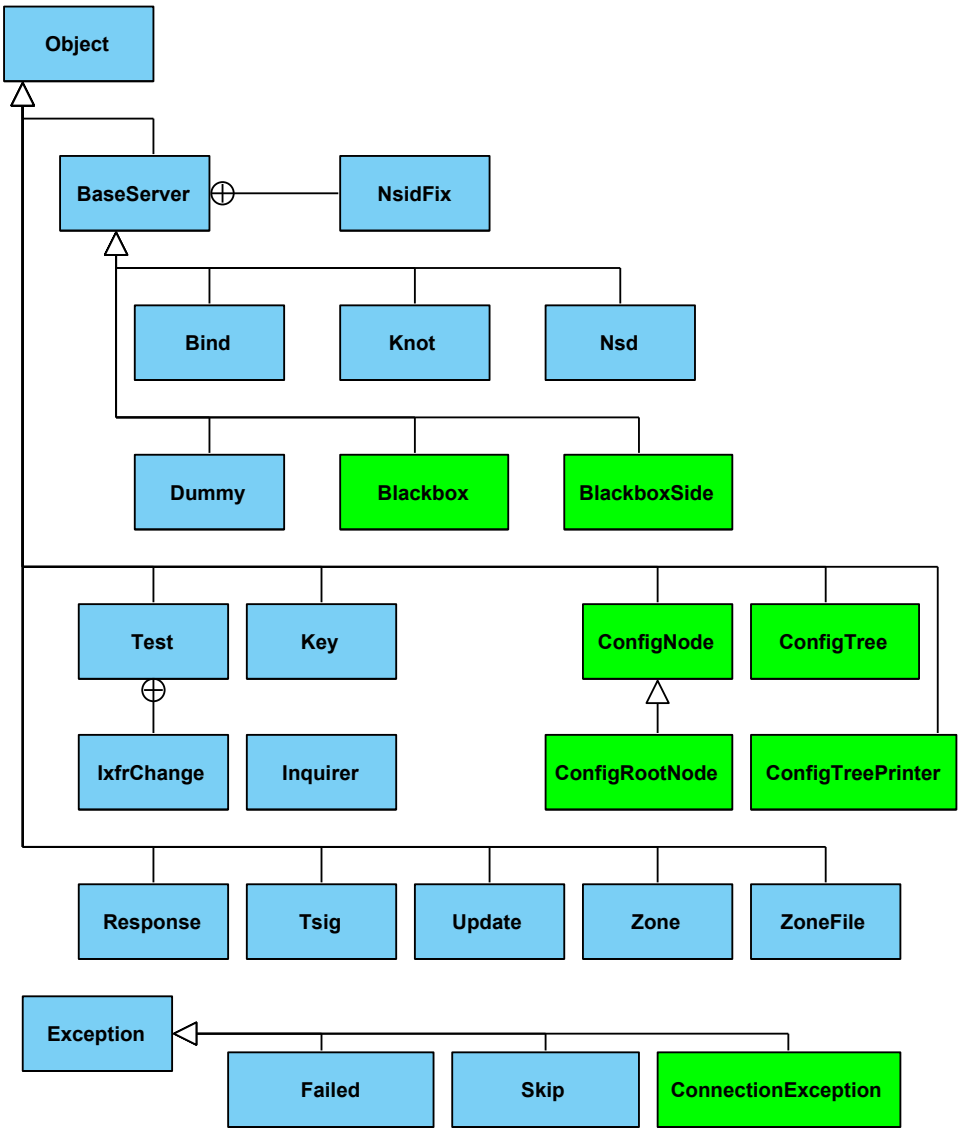
5.2 Flexibilita pri zápise testov

Zadaná požiadavka

Umožniť väčšiu flexibilitu pri zápise testov. Teda umožniť vykonanie špeciálnej operácie, ktorá neni priamo implementovaná frameworkom. Jedná sa hlavne o zmenu konfigurácie DNS serverov.

Riešenie požiadavky

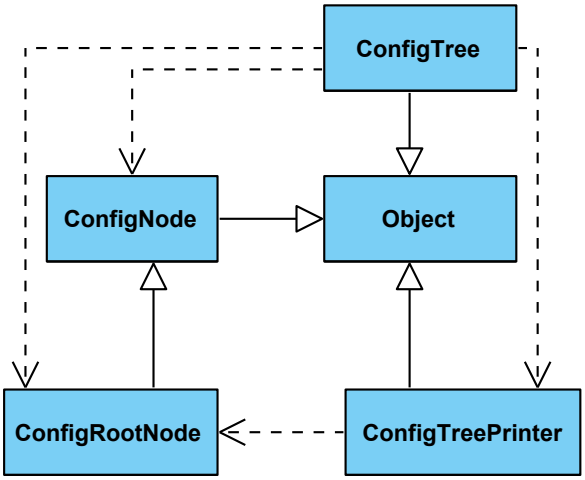
V pôvodnom frameworku neboli možné zmeny konfigurácie serverov z kódu testu. Všetky konfigurácie boli napevno zapísané v zdrojových súboroch frameworku, na základe ktorých boli konfiguračné súbory vygenerované a použité na konfiguráciu DNS serverov.



Obr. 5.2: Diagram tried upraveného frameworku

Na tieto účely bola vytvorená trieda, ktorá slúži na uloženie konfigurácie a nachádza sa v module `config.py`. Diagram tried modulu je vyobrazený na 5.3 a obsahuje nasledujúce triedy:

- `ConfigNode`
  Trieda reprezentuje uzol stromu. Atribúty uzlu sú meno, odkaz na rodičovský uzol a zoznam hodnôt a potomkov.
- `ConfigRootNode`
  Špeciálny typ uzlu. Trieda dedí z triedy obecného uzlu `ConfigNode`.
- `ConfigTreePrinter`
  Slúži na vytvorenie textovej reprezentácie stromu, ktorá slúži ako konfigurácia pre jednotlivé implementácie DNS serverov. Trieda obsahuje parametre slúžiace na formátovanie textového výstupu konfigurácie. Ide o formátovacie značky odsadenia, za-



Obr. 5.3: Diagram tried modulu `Config`

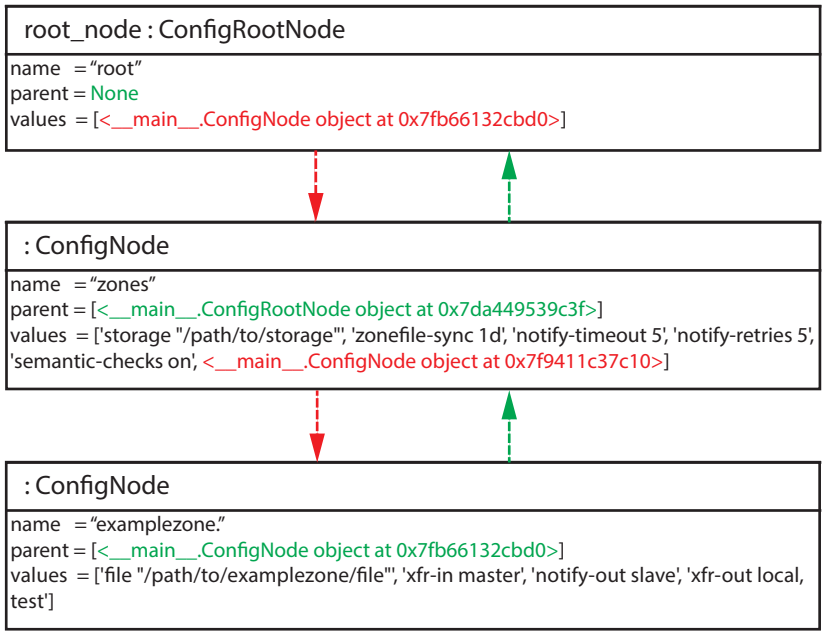
```
1  printer = ConfigTreePrinter()
2  config = ConfigTree(printer)
3
4  config.insert_to_root("zones", make_node=True)
5  config.insert_to_last("storage", "/path/to/storage")
6  config.insert_to_last("zonefile-sync 1d")
7  config.insert_to_last("notify-timeout 5")
8  config.insert_to_last("notify-retries 5")
9  config.insert_to_last("semantic-checks on")
10 config.insert_to_last("examplezone.", make_node=True)
11 config.insert_to_last("file", "/path/to/examplezone/file")
12 config.insert_to_last("xfr-in master")
13 config.insert_to_last("notify-out slave")
14 config.insert_to_last("xfr-out local, test")
```

Obr. 5.4: Príklad kódu vytvárajúceho konfiguračný strom

čiatku a konca uzlu a začiatku a konca atribútu. Jediná metóda `get_str` vytvára rekurzívnym priechodom textovú reprezentáciu stromu.

- `ConfigTree`
  Trieda reprezentujúca konfiguráciu servera. Zastrešuje všetky hlavné funkcie pre prácu s konfiguračným stromom. Obsahuje funkcie na pridanie prvkov atribút/uzol s voliteľnou hodnotou do koreňového alebo posledného pridaného uzla. Ďalej funkcie na vyhľadanie konkrétneho uzlu alebo atribútu a funkcie na vymazanie uzla zo stromu a získanie textovej reprezentácie konfigurácie. Na získanie textovej reprezentácie uloženej konfigurácie sa používa funkcia `get_config_str`.

Konfigurácia je reprezentovaná abstraktným dátovým typom obojsmerný viazaný strom (uzly obsahujú referenciu na otcovský uzol). Každý uzol stromu obsahuje zoznam. Tento zoznam obsahuje hodnoty priradené uzlu a uzly, ktoré sú jeho potomkami. Celá konfigurácia reprezentovaná stromom je teda uložená v koreňovom uzle - anglicky tzv. *root node*. Koreňový uzol neobsahuje odkaz na otcovský uzol. Na obrázku 5.4 je zobrazený príklad krátkej konfigurácie stromu.



Obr. 5.5: Príklad vytvoreného stromu spolu s jeho obsahom

```
1  zones {
2          storage "/path/to/storage";
3          zonefile-sync 1d;
4          notify-timeout 5;
5          notify-retries 5;
6          semantic-checks on;
7          examplezone. {
8                  file "/path/to/examplezone/file";
9                  xfr-in master;
10                 notify-out slave;
11                 xfr-out local, test;
12         }
13 }
```

Obr. 5.6: Vytvorený konfiguračný reťazec

Táto sekvencia príkazov (obr. 5.4) vytvorí v dátovej štruktúre strom spolu so zoznamami zobrazený na obrázku 5.5. Výsledkom tejto príkladovej konfigurácie je konfiguračný reťazec zobrazený na obrázku 5.7.

Každý objekt serveru obsahuje svoju vlastnú inštanciu konfiguračnej triedy, ku ktorej možno pristupovať a meniť jej obsah pomocou dostupného rozhrania triedy `ConfigTree`.

Výsledky a zhrnutie

Výsledkom je dynamická dátová štruktúra, ktorú možno meniť priamo z prostredia kódu testov. Konfiguráciu je možné viackrát zmeniť a vygenerovať aj počas priebehu testu. Príklad zmeny položky v konfigurácii je zobrazený na obr. 5.7. Cieľom je demonštratívne zmeniť v uzle „`zones`“ hodnotu atribútu „`notify-retries`“ z 5 na 10.

```
...
t = Test()

# vytvorenie objektu DNS servera Knot
knot = t.server("knot")

t.start()
...

# získanie zoznamu uzlov, ktoré obsahujú hľadaný reťazec
nodes = server.config_tree.get_attr("notify-retries.*")

# nájdi uzol s~menom "zones"
for node in nodes:
    if re.match("zones", node[0].name) is not None:
        zones_match_node = node

# zmena hodnoty atribútu "notify-retries" z~5 (predvolené) na 10
zones_node = zones_match_node[0]
index = zones_match_node[1]
zones_node.values[index] = "notify-retries 10"

# vygenerovanie nového konfiguračného súboru
server.gen_confile()

# opätovné načítanie konfigurácie serverom
server.reload()
...
```

Obr. 5.7: Príklad zmeny konfigurácie

5.3 Podpora ďalších implementácií DNS serverov

Zadaná požiadavka

Stávajúci framework podporuje implementácie BIND a Knot DNS. Cieľom je úprava frameworku do stavu, kedy bude možné jednoducho pridávať ďalšie implementácie vo forme modulov/tried, ktoré budú implementovať určité rozhranie a dediť z určitých abstraktných tried (základná trieda servera a konfigurácie). Cieľom teda bolo dosiahnutie stavu, kedy pridanie podpory pre ďalšiu implementáciu DNS servera:

- nevyžaduje žiadny zásah do implementácie frameworku,
- je jednoduché a modulárne,
- vyžaduje čo najmenšie úsilie na tvorbu nového modulu - maximum spoločne použiteľnej implementácie v bázovej triede servera.

Výsledný framework by mal podporovať minimálne tieto tri implementácie DNS serverov: BIND, Knot DNS a NSD.

Riešenie požiadavky

V pôvodnom frameworku boli všetky implementácie DNS serverov umiestnené v súbore `server.py`. Prvým krokom bolo rozčlenenie tohto modulu na menšie celky. Výsledkom ro-

zčlenenia bolo viacero modulov - `baseserver.py`, `config.py` a moduly pre DNS servery. Moduly pre DNS servery boli vyčlenené do samostatného priečinku `server`. Výsledné rozčlenenie je na obrázku 5.1.

Základným modulom sa stal modul `baseserver.py`, ktorý obsahuje abstraktnú (základnú) triedu `BaseServer`, z ktorej dedia všetky konkrétne implementácie serverov. Trieda obsahuje dve abstraktné metódy `get_config` a `listening`. Tieto metódy musí každá dediaca trieda definovať samostatne. Metóda `get_config` slúži na vygenerovanie textovej reprezentácie konfiguračných súborov pre dané implementácie DNS serverov a metóda `listening` sa používa na zistenie pripravenosti servera.

S metódou `get_config` súvisí modul `config.py`, ktorý bol popísaný v predchádzajúcej kapitole 5.2. Pomocou jeho triedy je možné servery konfigurovať presne na účely daných testov.

Moduly reprezentujúce implementácie serverov v samostatnom priečinku tvoria balíček `dnstest.server`. Pri importovaní tohto balíčka serverov sú importované všetky moduly na základe obsahu priečinka `server`.

Keďže obsah priečinka `server` sa môže meniť je potreba moduly načítavať dynamicky pri každom štarte frameworku. Importované sú všetky moduly Python-u, okrem modulu `__init__.py`. Framowork následne vyhľadáva implementované triedy v moduloch. Za triedu reprezentujúcu server určí tú triedu, ktorá má ako prvá zhodné meno s menom modulu (neberie sa do úvahy veľkosť písmen). Napríklad teda v module `exampleserver.py` bude za triedu reprezentujúcu server určená trieda s menom ako napríklad „ExampleServer“, „exampleServer“, a pod.

Každá nová implementácia DNS servera musí v module `params.py` definovať cestu k binárnemu súboru, ktorým sa DNS server spúšťa a cestu k binárnemu súboru, ktorý ovláda daný server. Pre server NSD boli pridané do modulu `params.py` príkazy zobrazené na obr. 5.8. Funkcia `get_binary` vyhľadá systémovú premennú zadaného názvu - prvý argument funkcie. Ak táto premená v systéme nie je definovaná, ako cesta k binárnemu súboru je použitá cesta uvedená ako druhý argument funkcie.

```
1 # KNOT_TEST_NSD - Nsd binary.
2 nsd_bin = get_binary("KNOT_TEST_NSD", "nsd")
3 # KNOT_TEST_NSDC - Nsd control binary.
4 nsd_ctl = get_binary("KNOT_TEST_NSDC", "nsdc")
```

Obr. 5.8: Príkazy definujúce binárne súbory DNS serveru NSD

Pre začlenenie implementácie DNS servera NSD bolo potreba implementovať viacero funkcií. Ako prvé boli implementované abstraktné metódy `get_config` a `listening` derivované z bázovej triedy. Ďalej boli implementované metódy:

- `_check_socket` – metóda bázovej triedy `BaseServer` nemohla byť použitá, pretože tá pri kontrole úspešnosti naviazania servera na danú adresu a port kontroluje počet ID procesov, ktoré sú na túto adresu naviazané. NSD však používa viacero procesov, ktoré sú na danú adresu naviazané.
- `start` – funkcia sa stará o spustenie implementácie servera.

- `ctl` – implementácia tejto metódy slúži na zasielanie príkazov serveru pomocou aplikácie vzdialeného ovládania.
- `reload` – slúži na zaslanie sekvencie príkazov, ktoré prinútia NSD server znovu načítať zóny zo zónových súborov.
- `flush` – funkcia pomocou programu na vzdialené ovládanie servera zasiela serveru príkaz „`patch`“, ktorý zapíše nezapísané zmeny do zónových súborov.
- `_get_pid` – slúži na získanie PID čísla procesu servera.
- `running` – upravená implementácia bázovej triedy. Táto implementácia pred pokusom o zistenie stavu DNS servera použije PID z funkcie `_get_pid`.
- `get_server_options` – vytvorí textovú reprezentáciu všetkých dostupných vlastností objektu servera.

Vzhľadom na to, že NSD vo verzii 3.2.18 podporuje iba niektoré algoritmy pre podpisovanie komunikácie (*TSIG*), bola do bázovej triedy pridaná premenná obsahujúca algoritmy podporované danou implementáciou DNS servera. NSD server túto hodnotu prepisuje podľa svojich vlastností. Tieto hodnoty dostupných algoritmov môžu byť použité pri výbere algoritmu pri komunikácii dvoch DNS serverov.

Výsledky a zhrnutie

Výsledkom je implementovaná podpora pre server NSD. Pre jeho začlenenie bolo potrebné prepísať v jeho triede zdedené funkcie základnej triedy `BaseServer`. Na testovanie boli použité upravené existujúce testy, aby zahŕňali testovanie implementácie NSD.

Príkladom je základný test dotazov na server NSD, ktorého odpovede sú následne porovnávané so serverom BIND. Test postupne testuje negatívne a pozitívne odpovede, dotazy typu ANY, CNAME a podobne. Priebeh testu je zobrazený na nasledujúcom obrázku 5.9.

```
1 23:29:40# KNOT TESTING SUITE 2015-04-14
2 23:29:40# Working directory /tmp/knottest-1429046980-8pqb5t
3 23:29:40# Test 'nsd'
4 23:29:45#  * case 'query':                OK
5 23:29:45# TEST CASES: 1, SUCCESS
```

Obr. 5.9: Výstup testovania NSD serveru

5.4 Paralelizácia testov

Zadaná požiadavka

Paralelizáciou sa treba zaoberať na úrovni frameworku, na úrovni testov alebo na úrovni jednotlivých príkazov testu. Pôvodný framework podporuje iba sériové vykonávanie zadaných testov a ich príkazov. Vhodným riešením by mohla byť paralelizácia na úrovni frameworku, pričom by bola použitá technika takzvaných „kontajnerov“ za použitia nástroja *Docker*.

Riešenie požiadavky

Počas riešenia práce bol pôvodný framework doplnený asociáciou CZ.NIC a na jednoduchú paralelizáciu bol použitý nástroj *Docker*[2]. Tento nástroj umožňuje vytvorenie kontajnerov, ktorých obsah je určený užívateľom.

Kontajner je prostredie, v ktorom sú spúšťané obrazy (anglicky *images*). Každý obraz obsahuje testované DNS servery, framework a všetok potrebný software. Obraz je možné vytvoriť pomocou nástroja `docker` a súboru `Dockerfile` alebo jednoducho spustiť daný kontajner a nástroj si najnovší obraz, ktorý bude spustený v kontajneri stiahne online z *Docker Hub* úložiska. Na tomto úložisku aktualizuje združenie CZ.NIC najnovší obraz pomenovaný `cznic/knot:tests-extra`. Všetky potrebné inštrukcie sú uvedené v súbore `README`.

V každom kontajneri je spustený rovnaký obraz. Pri spúšťaní kontajnera je možné zadať množinu testov, ktoré budú vykonané. Takto možno rozdeliť celú množinu testov na menšie podmnožiny, ktoré sú vykonané paralelne.

Výsledky a zhrnutie

Vďaka paralelizácii je možné každý jeden test spustiť paralelne, preto najdlhšia teoretická doba testovania by mala byť rovnaká ako dĺžka vykonania najdlhšie trvajúceho testu. Na výsledky testov paralelizácia nemá vplyv, pretože každý test (alebo množina testov) beží v kontajneri sériovo tak ako by bežala za normálnych okolností vo frameworku.

Zrýchlenie záleží čiste na dostupnom hardware, na ktorom sa paralelné testovanie vykonáva. Hlavným parametrom je výkon procesora, ktorý najvýraznejšie ovplivňuje dĺžku vykonávania testov. Príklad paralelizácie a jej výsledky sú uvedené v kapitole s príkladovými testami - kap. 6.5.

5.5 Vytvorenie *dummy* serveru

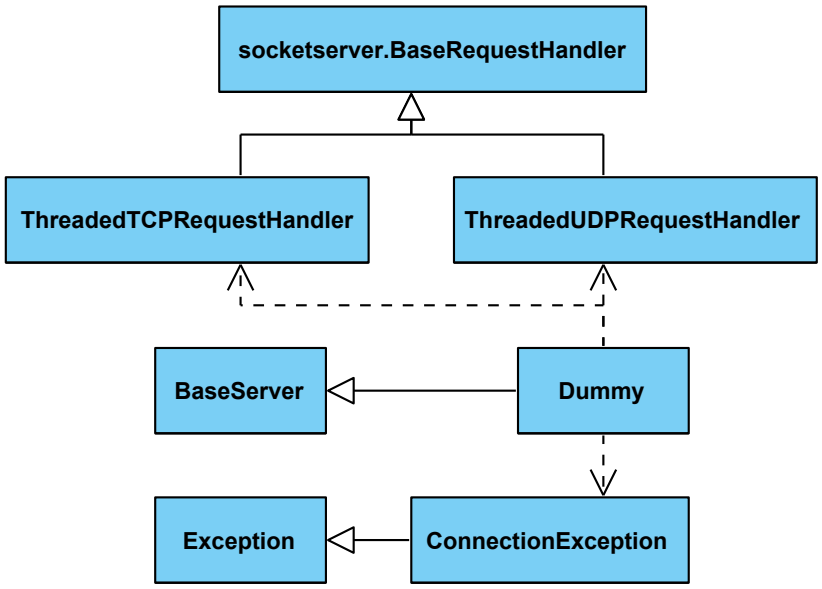
Zadaná požiadavka

V niektorých prípadoch testovania je potreba posielať predpripravené odpovede testovaným serverom. Pôvodný framework túto možnosť podporuje iba nízkoúrovňovým prístupom ku knižnici `dnspython` a manuálnym vytváraním odpovedí.

Riešenie požiadavky

Riešením je *Dummy* server, ktorý slúži ako jednoduchý server odpovedajúci pripravenými odpoveďami. Za týmto účelom bola vytvorená trieda `Dummy` v module `dummy.py`. Keďže tento server sa z pohladu frameworku používa rovnako ako ostatné implementácie DNS serverov, je umiestnený v priečinku `server`.

[2] Nástroj používa softwarové kontajnery, ktoré obsahujú požadovaný software. `https://www.docker.com/whatisdocker/`



Obr. 5.10: Diagram tried *dummy* serveru

Server je implementovaný ako viacvláknový. Za týmto účelom využíva triedy `ThreadingTCPServer` a `ThreadingUDPServer` definované v Python knižnici `socketserver`. Diagram tried serveru je zobrazený na obrázku 5.10. Pre oba protokoly TCP a UDP je vytvorený samostatný server, ktorý naslúcha na danom porte a pri prijatí dát vytvára nové „pracovné“ vlákno, ktoré dáta spracuje. Pracovné vlákno zavolá funkciu `handle`, ktorá je definovaná v triedach `ThreadedTCPRequestHandler` a `ThreadedUDPRequestHandler` podľa typu protokolu. Tá po základnom predspracovaní dát vytvorí bitové pole (typ `bytearray`) a zavolá funkciu, ktorú na tento účel priradil užívateľ. Táto funkcia je priradená do premennej serveru `user_handler`. Výsledné dáta spracované užívateľskou funkciou sú zaslané naspäť odosielateľovi.

Posielanie pripravených odpovedí v skutočnosti nie je iba odosielanie pripravených odpovedí. Nejedná sa v pravom slova zmysle o pripravené odpovede, ktoré by boli pripravené ešte pred štartom servera. Odpovede sú tvorené dynamicky. Pre tento účel je užívateľ povinný zadať funkciu, ktorá ako vstupný parameter dostane pole bytov a vráti taktiež pole bytov. Obsah návratových dát je plne v réžii užívateľa.

Server je implementovaný v triede `Dummy` a ako každý server, dedí z bázovej triedy. Aby bol zabezpečený jeho chod prepisuje nasledujúce funkcie bázovej triedy:

- `_check_socket` - určuje, či je daný port priradený serveru
- `get_config` - vracia konfiguráciu servera; server nepoužíva žiadnu konfiguráciu a teda vracia prázdny reťazec
- `listening` - vracia pravdivostnú hodnotu `True` alebo `False` na základe toho, či server naslúcha na danom porte na TCP aj UDP spojenia
- `running` - kontroluje či server beží
- `start` - základné nakonfigurovanie adresy a portu serveru a spustenie hlavných vlákien pre TCP a UDP spojenia

```
22:54:25# KNOT TESTING SUITE 2015-04-14
22:54:25# Working directory /tmp/knottest-1429044865-dxxi6s
22:54:25# Test 'dummy'
22:54:26#  * case 'query':                OK
22:54:26# TEST CASES: 1, SUCCESS
```

Obr. 5.11: Výstup testovania *dummy* serveru

```
2015-04-14T22:54:25 Starting main threads for TCP and UDP requests
2015-04-14T22:54:25 Server main threads running
2015-04-14T22:54:25 Listening on ::1@18046
2015-04-14T22:54:25 TCP data from ::1@45607
2015-04-14T22:54:25 UDP data from ::1@33909
2015-04-14T22:54:25 Shutting down main threads for TCP and UDP requests
2015-04-14T22:54:26 Server main threads shut down
```

Obr. 5.12: Výstup logovania *dummy* serveru

- `stop` - zastavenie hlavných vlákien serveru a ich potomkov (vlákna vytvorené hlavnými vláknami)

Výsledky a zhrnutie

Výsledkom je jednoduchý viacvláknový server, ktorý upravuje prijaté dáta pomocou užívateľom definovanej funkcie. Testovanie prebiehalo iba pomocou jedného testu, ktorým bola overená komunikácia a správnosť doručenia odpovede príjemcovi. Test teda obsahuje iba vytvorenie dvoch inštancií *dummy* serverov a dotazy na jednotlivé servery s následným porovnaním ich odpovedí. Užívateľská funkcia v tomto prípade iba zmenila dotaz na odpoveď a nastavila požadované príznaky správy tak, aby vyhovovali očakávaniam príjemcu. Priebeh testu je zobrazený na obrázku 5.11.

Dummy server implementuje aj podporu pre jednoduché logovanie udalostí, táto možnosť je implicitne povolená pomocou príznaku *dummy* serveru `log_traffic`. Výstup logovania je na obrázku 5.12.

5.6 *Blackbox*

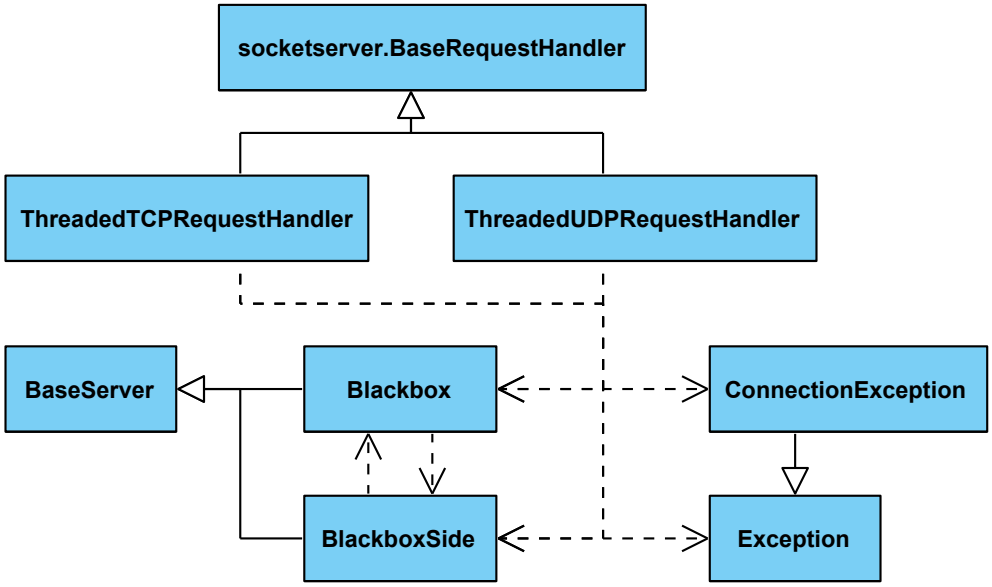
Zadaná požiadavka

Pôvodný framework je potrebné doplniť o implementáciu chýbajúceho prvku *Blackbox*. Tento prvok by mal byť umiestnený medzi dvoma DNS servermi a ich komunikácia by mala prebiehať prostredníctvom tohto prvku. Užívateľ by mal mať možnosť pristupovať k tejto komunikácii a ľubovolne s ňou manipulovať. To znamená, že dáta môže ľubovolne upraviť, vymazať, oneskoriť a podobne. *Blackbox* by teda mohol slúžiť na simulovanie chýb pri prenose ako napríklad vytváranie chýb na úrovni bitov, vytváranie latencie linky medzi servermi a podobne.

Riešenie požiadavky

Požiadavka *Blackbox* predstavuje vytvorenie objektu, ktorý simuluje chyby pri prenose medzi dvoma DNS servermi.

Základom *blackbox*-u je implementácia *dummy* serveru. Taktiež ide o viacvláknový TCP a UDP server používajúci Python modul `socketserver`. *Blackbox* je implementovaný ako proxy server, ktorý pri prijatí dát tieto sprístupní užívateľovi. Následne po dokončení úprav užívateľom sú dáta preposlané ich zadanému príjemcovi. Diagram tried *blackbox* prvku je na obrázku 5.13.

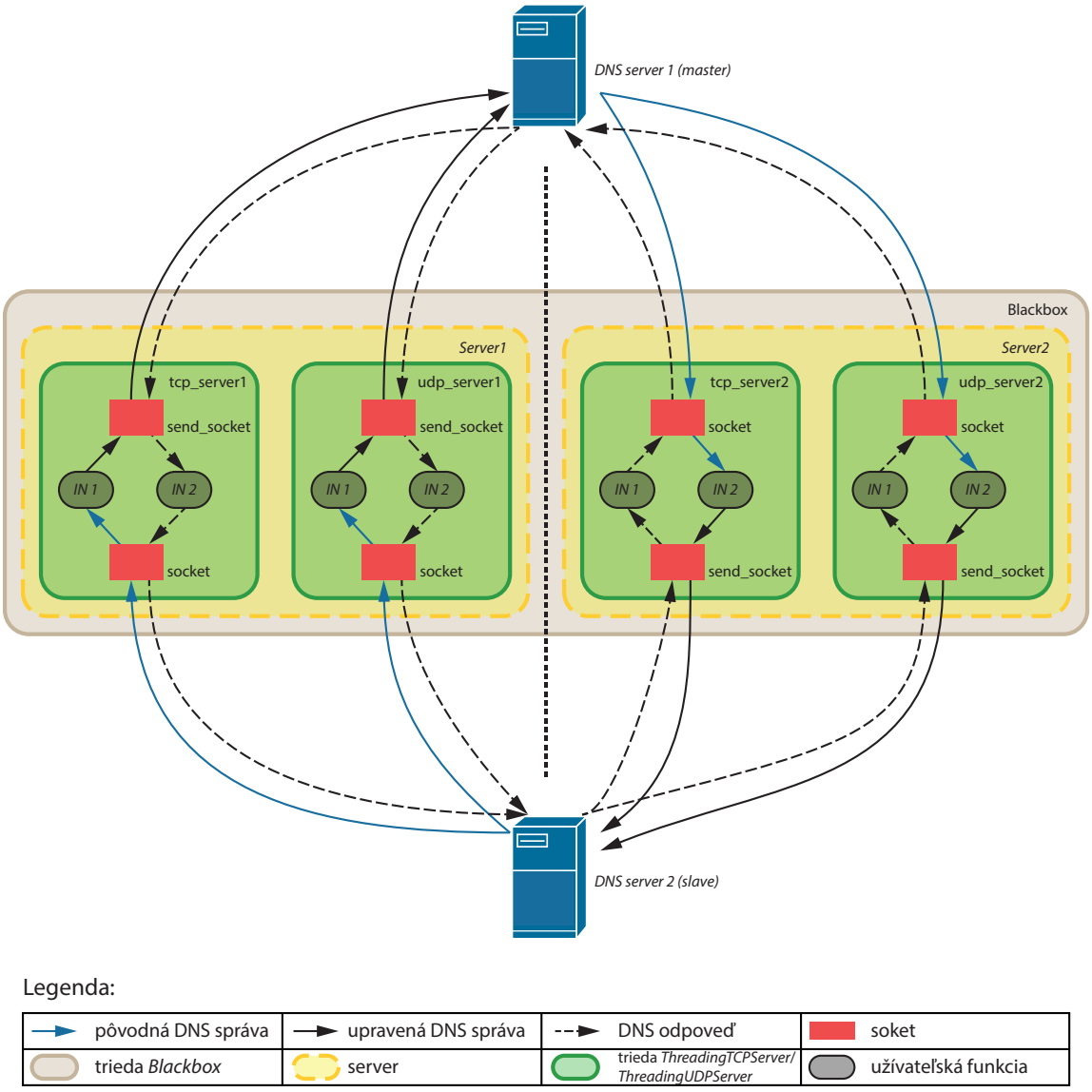


Obr. 5.13: Diagram tried *blackbox* serveru

Pre lepšie pochopenie vykonaných zmien, ktoré budú popisované ďalej v tejto kapitole je potrebné vysvetliť štruktúru a prepojenie *blackbox* serveru s DNS servermi, ktoré spája. Schéma pripojenia a hierarchie prvku *blackbox* je zobrazená na nasledujúcom obrázku 5.14.

Podľa schémy 5.14 a diagramu tried 5.13 je zrejmé, že trieda `Blackbox` sa obsahuje inštancie nasledujúcich tried (popísané iba dôležité triedy):

- `BlackboxSide` - trieda reprezentuje jeden „server“ – jednu serverovú stranu *blackbox*-u. Ide o „falošný“ server, ktorý dedí z bázovej triedy `BaseServer`. Objekty tejto triedy slúžia pre spájané DNS servery na získanie komunikačných vlastností proxy servera. Ide hlavne o IP adresu a port *blackbox*-u.
- `ThreadingTCPServer` a `ThreadingUDPServer` - triedy modulu `socketserver`. Zaobstarávajú príjem komunikácie na UDP a TCP portoch a vytvárajú nové pracovné vlákna na spracovanie komunikácie.
- `Thread` - predstavuje štyry hlavné vlákna, v ktorých sú spustené triedy `ThreadingTCPServer` a `ThreadingUDPServer`.
- `BaseServer` - bázová trieda pre implementácie serverov. Slúži na získanie konfigurácie serverov napojených na proxy server.



Obr. 5.14: Schéma pripojenia a hierarchie prvku *blackbox*

Vytvorenie objektu triedy `Blackbox` je rovnaké ako u iných serverov cez rozhranie objektu triedy `Test`. Konfigurovať stačí iba dva atribúty objektu:

- atribút `log_traffic` - na základe pravdivostnej hodnoty obsahu atribútu proxy server (ne)vykonáva logovanie stručných informácií o prichádzajúcich dátach.
- atribúty `server1_ingress_user_handler` a `server2_ingress_user_handler` - ide o atribúty, do ktorých užívateľ priradí funkcie, ktorým majú byť predané dáta pred vstupom na `server1`, respektíve `server2`.

O následné prepojenie *blackbox*-u s oboma servermi sa postará funkcia `link` triedy `Test`. Zadaním objektu triedy `Blackbox` ako parametra sa vo funkcii vytvorí prepojenie `server1`→`Blackbox` a `Blackbox`→`server2`.

Testovanie DNS serverov pomocou frameworku prebieha väčšinou na localhoste. DNS servery umožňujú v konfiguráciách určiť port na ktorom čakajú príchodziu komunikáciu. Toto číslo portu však nie je použité ak DNS server zahajuje komunikáciu s iným serverom/klientom. Vďaka tomu teda nie je možné na základe čísla portu, z ktorého prišli dáta, zistiť, ktorý server dáta poslal a teda nemožno určiť ani to, ktorému serveru dáta poslať ďalej. Z tohto dôvodu je *Blackbox* tvorený štyrmi servermi. Každému serveru na strane proxy serveru sú priradené dva servery – jeden server pracujúci s TCP protokolom a druhý s UDP protokolom, oba pripojené na rovnakú IP adresu a port. Oba servery pripojené na proxy server teda dostanú iné číslo portu proxy serveru. Vďaka tomu je možné rozoznať od ktorého serveru dáta prišli a kam ich treba ďalej poslať.

Prijaté dáta sú vždy pred preposlaním cieľovému serveru predané do užívateľskej funkcie. Do užívateľskej funkcie vstupujú cez parametre dáta vo forme poľa bytov, objekt serveru ktorému sú dáta určené a textový reťazec „tcp“ alebo „udp“, podľa toho pomocou ktorého transportného protokolu boli prijaté. Užívateľ upraví alebo inak pozmení dáta, poprípade pozdrží dáta na určitý čas. Dáta vrátené užívateľskou funkciou musia byť opäť typu triedy `bytearray`. Tieto sú však poslané príjemcovi na druhej strane (neposielajú sa naspäť odosielateľovi ako v prípade *dummy* serveru).

V momente prijatí dát sa soket spojený s pôvodcom (iniciátorom) komunikácie nezatvára. Otvorí sa nový soket pre komunikáciu s druhým serverom, cez ktorý sú odoslané upravené dáta vrátené užívateľskou funkciou. Po prijatí dát z druhého servera sú tieto zaslané pôvodcovi (iniciátorovi). Cyklus preposielania a upravovania prechádzajúcej komunikácie v prípade protokolu TCP prebieha pokiaľ neni detekované uzatvorenie spojenia jedným zo serverov. V prípade protokolu UDP prebehne cyklus iba raz, keďže UDP je bezstavový protokol.

V porovnaní s ostatnými DNS servermi v rámci frameworku a *dummy* serverom je *blackbox* špeciálny server. Pre jeho začlenenie do frameworku bolo potrebné vykonať nasledovné zmeny:

1. Pretože sa jedná o špeciálny server s mnohými odlišnosťami oproti ostatným „obyčajným“ serverom, je implementácia *blackbox*-u umiestnená mimo ostatných serverových implementácií v priečinku `./tests-extra/tools/dnstest` v module `blackbox.py`.
2. Na základe obrázkov 5.14 a 5.13 je zrejmé, že trieda `Blackbox` obsahuje dva objekty triedy `BlackboxSide`. Tieto objekty sa nevytvárajú rovnako ako ostatné servery, preto trieda `Test` bola rozšírená o funkciu `_create_blackboxside`. Tá vytvorí objekt triedy `BlackboxSide` a jeho atribúty (zdedené z abstraktnej triedy `BaseServer`) naplní hodnotami z nadradeného objektu triedy `Blackbox`.
3. Pri vytváraní *Blackbox* serveru cez rozhranie triedy `Test` bola opätovne upravená funkcia `Test.server`. Tá najprv vytvorí objekt triedy `Blackbox` a následne priradí tomuto objektu dva objekty triedy `BlackboxSide` pomocou vyššie spomínanej funkcie `Test._create_blackboxside`.
4. Pre jednoduchšie používanie prvku *blackbox* bola upravená ďalšia funkcia triedy `Test`, konkrétne funkcia `link`. Úpravou bolo pridanie ďalšieho parametru `blackbox`, ktorého uvedením užívateľ žiada vloženie daného objektu triedy `Blackbox` medzi dané dva DNS servery. Funkcia sa následne postará o prepojenie všetkých troch objektov.

```
#!/usr/bin/env python3

from dnstest.test import Test
from dnstest.utils import *
import dns.message

def my_handler(msg):
    return msg

t = Test(tsig=False)

master = t.server("bind")
slave = t.server("knot")
zones = t.zone("flags.")

blackbox = t.server("blackbox")
blackbox.server1_ingress_user_handler = my_handler
blackbox.server2_ingress_user_handler = my_handler
blackbox.log_traffic = True

t.link(zones, master, slave, blackbox=blackbox)

t.start()
...
t.end()
```

Obr. 5.15: Základná konfigurácia testu pre *blackbox*

Výsledky a zhrnutie

Výsledkom je jednoduchý viacvláknový *Blackbox* – DNS proxy server, ktorý spája dva ľubovolné servery frameworku. Jednoduché logovanie proxy serveru je rovnaké ako u *dummy* serveru. anglickyBlackbox je jednoducho použiteľný a vyžaduje minimálnu konfiguráciu. Na obrázku 5.15 je príklad základnej konfigurácie testu s použitím triedy `Blackbox`.

5.7 Vytvorenie exemplárnych testov

Zadaná požiadavka

Nad upraveným frameworkom je potreba vytvoriť základnú množinu testov pre demonštráciu požadovaných funkcií a vykonaných zmien. Ide o základné testy pre prenos zón, dotazy a podobne.

Riešenie požiadavky

Na všetky testovateľné úpravy boli vytvorené základné testy demonštrujúce ich funkčnosť. Všetky tieto testy, ich význam, ich výstupy a iné bude popísané v nasledujúcej kapitole 6.

Kapitola 6

Príkladové testy

Posledným bodom zadania bolo vytvorenie testov demonštrujúcich funkčnosť upraveného frameworku. Boli vytvorené základné testy demonštrujúce funkčnosť vytvorených prvkov a nových vlastností frameworku. Nasledujúce podkapitoly popisujú niektoré vytvorené testy a ich výsledky.

Všetky testy boli vykonávané na osobnom počítači s hardwarovými parametrami:

- Intel Core 2 Duo 2,4 GHz,
- 4GB RAM,

a nasledujúcimi softwarovými parametrami:

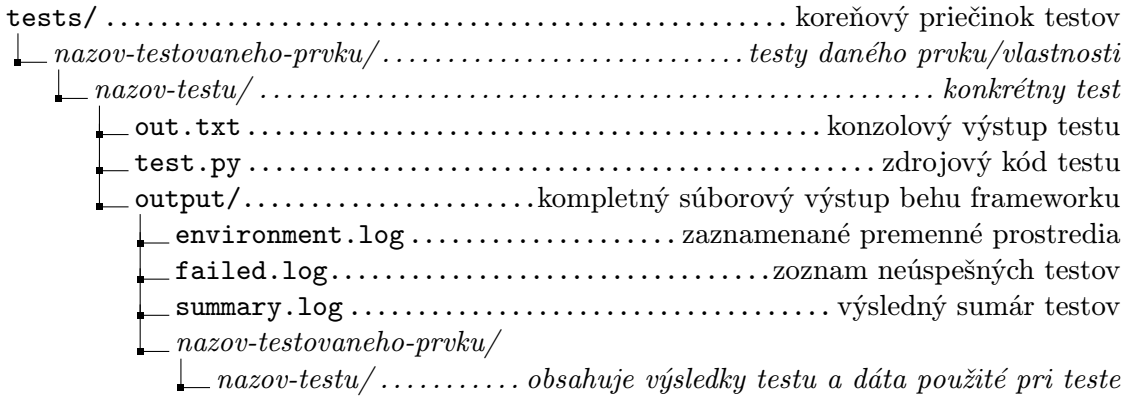
- operačný systém Fedora release 20 (Heisenbug),
- Knot DNS 1.6.3,
- BIND 9.9.4-P2-RedHat-9.9.4-18.P2.fc20 (Extended Support Version),
- NSD 3.2.18,
- Docker 1.5.0,
- Python 3.3.2.

Samotné testy spolu s testovacími dátami a výsledkami sú umiestnené na priloženom CD nosiči v priečinku `tests`. Štruktúra priečinku je popísaná na obrázku 6.1. Táto štruktúra je použitá pre všetky výsledky vykonaných testov okrem paralelizácie. Priečinok `output` obsahuje presne rovnakú štruktúru ako je popísaná v kapitole 4.3.1 pri popisovaní výstupu frameworku.

Znovuspustenie vytvorených testov, ktoré boli vytvorené za účelom otestovania nových vlastností frameworku je možné spustiť bash skriptom `run.sh`.

6.1 Flexibilita pri zápise testov

Keďže týmto bodom medzi požiadavkami bola myslená konfigurácia bežiacich DNS serverov, je v tejto časti testovania vykonaný iba jeden demonštratívny test. Tento test vykoná



Obr. 6.1: Štruktúra priečinku s testami

zmenu položky v konfigurácii DNS serveru Knot DNS. Cieľom je zmeniť v uzle „`zones`“ hodnotu atribútu „`notify-retries`“ z 5 na 10. Test je zobrazený na obrázku (obr. 6.2).

```
#!/usr/bin/env python3
''' Config change example '''

from dnstest.test import Test
import re

t = Test(tsig=False)
server = t.server("knot")

t.start()
t.sleep(2)

# change config
nodes = server.config_tree.get_attr("notify-retries.*")
# change "notify-retries" attribute value from 5 (default) to 10
for node in nodes:
    if re.match("zones", node[0].name) is not None:
        zones_match_node = node

zones_node = zones_match_node[0]
index = zones_match_node[1]
zones_node.values[index] = "notify-retries 10"

# generate new config file
server.gen_confile()
# reload new configuration
server.reload()

t.end()
```

Obr. 6.2: Príklad zmeny konfigurácie

Po začatí testu obsahuje konfiguračný súbor v uzle „`zones`“ atribút „`notify-retries 5;`“. Po zmene konfigurácie a vygenerovaní konfiguračného súboru je obsah uzlu „`zones`“ zobrazený na obr. 6.3. Tento súbor už obsahuje novú upravenú hodnotu atribútu „`notify-retries 10;`“.

```
zones {
        storage "/tmp/knottest-1431857525-86pswr/config/change/knot1";
        zonefile-sync 1d;
        notify-timeout 5;
        notify-retries 10;
        semantic-checks on;
}
```

Obr. 6.3: Zmenená časť konfiguračného súboru

6.2 *Dummy* server

Dummy server slúži ako jednoduchý TCP/UDP server, ktorý prijíma dáta a po ich úprave užívateľom ich posiela naspäť odosielateľovi. Jedným z testov tohto serveru bolo vytvorenie odpovede na prijaatú DNS správu NOTIFY. Podľa RFC 1996 [3] sa vytvorí odpoveď z prijatej NOTIFY správy iba pridaním QR príznaku. Test je uvedený na obrázku 6.4.

```
#!/usr/bin/env python3
''' Notify message from Knot to Dummy server. '''

from dnstest.utils import *
from dnstest.test import Test
import dns.message

def my_handler(data, protocol):
    # make DNS message and set QR flag
    response_msg = dns.message.from_wire(data)
    response_msg.flags |= dns.flags.QR

    print("\nProtocol:"+protocol)
    print(response_msg)

    # return byteArray data
    return response_msg.to_wire()

t = Test(tsig=False)
master = t.server("knot")
dummy = t.server("dummy")
dummy.DIG_TIMEOUT = 5
dummy.user_handler = my_handler
zone = t.zone("flags.")
t.link(zone, master, dummy)

t.start()
t.sleep(2)
master.stop()
master.start()
t.sleep(2)
t.end()
```

Obr. 6.4: Test *Dummy* serveru

Po štarte je master server po dvoch sekundách reštartovaný, aby mohol zaslať NOTIFY správu *Dummy* serveru. Na základe logu Knot DNS serveru z obrázku 6.5 a logu *Dummy* serveru z obrázku 6.6 môžeme vidieť, že v čase 14:17:44 kedy po prvýkrát Knot DNS server posiela správu NOTIFY ešte *Dummy* server nebeží. Druhýkrát server Knot DNS posiela NOTIFY správu v čase 14:17:48. Táto správa je prijatá *Dummy* serverom a predaná do užívateľskej funkcie.

```
2015-05-17T14:17:44 info: Knot DNS 1.6.0 starting
...
2015-05-17T14:17:44 warning: [flags] NOTIFY, outgoing, ::1@12037: failed (connection
    refused)
...
2015-05-17T14:17:48 info: shutting down
2015-05-17T14:17:48 info: Knot DNS 1.6.0 starting
...
2015-05-17T14:17:48 info: [flags] NOTIFY, outgoing, ::1@12037: serial 2010111213
...
2015-05-17T14:17:52 info: shutting down
```

Obr. 6.5: Časť logovacieho výstupu Knot DNS serveru

```
2015-05-17T14:17:46 Starting main threads for TCP and UDP requests
2015-05-17T14:17:46 Server main threads running
2015-05-17T14:17:46 Listening on ::1@12037
2015-05-17T14:17:48 TCP data from ::1@46646
2015-05-17T14:17:52 Shutting down main threads for TCP and UDP requests
2015-05-17T14:17:52 Server main threads shut down
```

Obr. 6.6: Logovací výstup *Dummy* serveru

Užívateľská funkcia po prijatí dát tieto prekonvertuje na objekt `dns.message.Message`. Následne je pridaný príznak QR a obsah celej správy je vypísaný (obr. 6.7) spolu s pridaným prvým riadkom, ktorý označuje transportný protokol použitý pre prijatie dát. Na koniec je správa prekonvertovaná naspäť na pole bytov a je odoslaná naspäť odosielateľovi.

```
Protocol:tcp
id 1607
opcode NOTIFY
rcode NOERROR
flags QR AA
;QUESTION
flags. IN SOA
;ANSWER
flags. 3600 IN SOA dns1.flags. hostmaster.flags. 2010111213 10800 3600 1209600 7200
;AUTHORITY
;ADDITIONAL
```

Obr. 6.7: Výstup *Dummy* serveru

6.3 *Blackbox*

Test s názvom „`axfr_bind_knot`“ ukazuje použitie prvku *blackbox* použitého na prepojenie serverov, ktoré následne vykonajú zónový prenos typu *AXFR*. Zdrojový kód testu je zobrazený na obrázku 6.8.

```
1  #!/usr/bin/env python3
2
3  ''' AXFR from Bind to Knot via Blackbox '''
4
5  from dnstest.test import Test
6  from dnstest.utils import *
7
8  def my_handler(data, to_server, protocol):
9      return data
10
11 t = Test(tsig=False)
12
13 master = t.server("bind")
14 slave = t.server("knot")
15 zones = zone = t.zone("flags.")
16
17 blackbox = t.server("blackbox")
18 blackbox.server1_ingress_user_handler = my_handler
19 blackbox.server2_ingress_user_handler = my_handler
20
21 t.link(zones, master, slave, blackbox=blackbox)
22
23 t.start()
24 t.sleep(2)
25 master.stop()
26 master.start()
27 t.sleep(5)
28 t.end()
```

Obr. 6.8: *AXFR* prenos zón cez `Blackbox`

Na začiatku testu je definovaná užívateľská funkcia `my_handler`, ktorá prijaté dáta nijak nemení, iba ich vráti naspäť. Nasleduje vytvorenie objektu triedy `Test`, ktorý zastrešuje rozhranie pre testovanie. Pomocou tohto objektu sú následne vytvorené objekty serverových tried `Bind`, `Knot` a `Blackbox`. V teste je použitá jednoduchá testovacia zóna `flags.`. Prepojenie je vykonané pomocou funkcie `link`, ktorá medzi master a slave server vloží *blackbox*. Pod prepojením sa chápe viazanosť serverov na základe uvedenej zóny.

Po zahájení testu sú spustené všetky tri servery. Čakanie na spustenie serverov je vo frameworku implicitne nastavené, no pre istotu dostatku času je pridaný čakací interval dve sekundy. Po uplynutí doby čakania je master server zastavený a znovu spustený. Deje sa tak kvôli tomu, aby bol zaručený beh slave servera pri spúšťaní master servera. Pri spúšťaní master servera totižto dochádza k posielaniu DNS správy NOTIFY, ktorá informuje slave servery o aktualizácii dát zóny. Po prijatí tejto správy slave server odpovie „potvrdzovacou“ DNS správou príjem NOTIFY správy a zaháji zónový prenos.

Pri konkrétnom behu testu boli pridelené porty nasledovne:

- Bind (master): počúva na porte 18009 a pozná port blackboxu 18011,
- Blackbox: počúva na portoch 18011 a 18017,

```
11-May-2015 13:07:00.325 starting BIND 9.9.4...
...
11-May-2015 13:07:00.346 zone flags/IN: sending notifies (serial 2010111213)
11-May-2015 13:07:02.492 client 127.0.0.1#43954 (flags): transfer of 'flags/IN':
    AXFR started
11-May-2015 13:07:02.492 client 127.0.0.1#43954 (flags): transfer of 'flags/IN':
    AXFR ended
11-May-2015 13:07:06.570 shutting down
...
11-May-2015 13:07:06.593 starting BIND 9.9.4...
...
11-May-2015 13:07:06.610 zone flags/IN: sending notifies (serial 2010111213)
11-May-2015 13:07:13.597 shutting down
...
```

Obr. 6.9: Logovací výstup servera Bind

```
2015-05-11T13:07:00 Starting main threads for TCP and UDP requests
2015-05-11T13:07:00 Server main threads running
2015-05-11T13:07:00 Listening on 127.0.0.1@18017 and 127.0.0.1@18011 for UDP and TCP
2015-05-11T13:07:00 UDP data from 127.0.0.1@39557 (server='bind1') to 127.0.0.1
    @18011
2015-05-11T13:07:02 TCP data from 127.0.0.1@57644 (server='knot1') to 127.0.0.1
    @18017
2015-05-11T13:07:06 UDP data from 127.0.0.1@18891 (server='bind1') to 127.0.0.1
    @18011
2015-05-11T13:07:06 TCP data from 127.0.0.1@57646 (server='knot1') to 127.0.0.1
    @18017
2015-05-11T13:07:13 Shutting down main threads for TCP and UDP requests
2015-05-11T13:07:14 Server main threads shut down
```

Obr. 6.10: Štandardný logovací výstup servera Blackbox

- Knot (slave): počúva na porte 18015 a pozná port blackboxu 18017.

Na obrázku 6.9 je zobrazený výpis servera Bind. Vo výpise vidieť, že server bol spustený dvakrát, dvakrát načítal zónový súbor `flags.`, dvakrát zaslal DNS správu NOTIFY a stihol vykonať zónový transfer už pri prvom spustení (čas 13:07:02).

Štandardný logovací výpis *blackbox* serveru na obrázku 6.10 ukazuje postupnosť komunikácie, ktorá cez neho prebehla. V čase 13:07:00 môžeme vidieť prvý pokus o zaslanie DNS správy NOTIFY zo serveru Bind na server Knot. Na základe logovacích výstupov serveru Bind (obr. 6.9), serveru Knot (obr. 6.12) a chybového logovacieho výstupu serveru Blackbox (obr. 6.11) môžeme vidieť, že prvý pokus o doručenie správy NOTIFY nebol úspešný.

AXFR prenos zóny však bol úspešne uskutočnený v čase 13:07:02. Blackbox zaznamenal iniciáciu TCP spojenia od serveru Knot na porte 57644. V logu serveru Bind sa v tomto čase objavuje začiatok prenosu a koniec prenosu zóny. Tak isto je tomu aj v logu serveru

```
2015-05-11T13:07:00 UDP handler error 'Traceback (most recent call last):
  File "/media/Data/Dropbox/Škola/DP/knot-dns/tests-extra/tools/dnstest/blackbox.py
    ", line 221, in handle data = orig_sender[1](send_to_socket.recv(self.server.
    max_packet_size), orig_sender[0], "udp") ConnectionRefusedError: [Errno 111]
    Connection refused'
```

Obr. 6.11: Chybový logovací výstup servera Blackbox

Knot.

```
2015-05-11T13:07:02 info: Knot DNS 1.6.0 starting
2015-05-11T13:07:02 info: binding to interface '127.0.0.1@18015'
...
2015-05-11T13:07:02 info: [flags] AXFR, incoming, 127.0.0.1@18017: starting
2015-05-11T13:07:02 info: [flags] AXFR, incoming, 127.0.0.1@18017: finished, serial
    0 -> 2010111213, 0.00 seconds, 1 messages, 2628 bytes
2015-05-11T13:07:06 info: [flags] NOTIFY, incoming, 127.0.0.1@60661: received serial
     2010111213
2015-05-11T13:07:06 info: [flags] refresh, outgoing, 127.0.0.1@18017: zone is up-to-
    date
...
2015-05-11T13:07:14 info: shutting down
```

Obr. 6.12: Logovací výstup servera Knot

Pri druhom štarte master serveru Bind tento znova posiela DNS správu NOTIFY (čas 13:07:06). Slave server Knot má však aktuálny obsah zónového súboru a preto znova nezačína prenos zóny. Zónový prenos bol teda úspešný aj pri použití prvku *Blackbox*.

6.4 Rozšíriteľnosť o implementácie DNS serverov

Framework bol upravený aby bol schopný čo najjednoduchšie rozšíriť svoje možnosti o testovanie nových implementácií DNS serverov. Oproti pôvodnému frameworku bol framework rozšírený o podporu testovania implementácie DNS servera NSD. Nasledujúci test demonštruje *IXFR* zónové prenos medzi Knot DNS (master) a NSD (slave) servermi.

Zdrojový kód testu je zobrazený na obrázku 6.13. Najprv sú vytvorené objekty DNS serverov, následne je vygenerovaných päť náhodných zón, každá s počtom päťdesiat záznamov. Servery sú následne „spojené“ vytvorenými zónami.

Následne sa pomocou funkcie `zones_wait` získajú aktuálne sériové čísla zón zo SOA záznamov. V cykle sa potom vykonávajú nasledujúce kroky:

- každá zóna master serveru sa upraví a zvýši sa jej sériové číslo v SOA zázname,
- master server znova načíta zónové súbory,
- pomocou funkcie `zones_wait` sa znova počká na dokončenie prenosu zón a zistia sa aktuálne sériové čísla zón.

Pred koncom testu je zadaný príkaz serveru NSD, ktorý spôsobí, že NSD server zapíše zóny do zónových súborov na disk. Výsledkom sú aktuálne zónové súbory na strane NSD (slave) serveru.

6.5 Paralelizácia

Príklad paralelizácie bol vykonaný nad aktuálnym obrazom `cznic/knot:tests-extra`, ktorý je dostupný na *Docker Hub* úložisku. Pre jeho stiahnutie a zároveň jeho spustenie stačí po nainštalovaní *Docker* nástroja zadať príkaz:

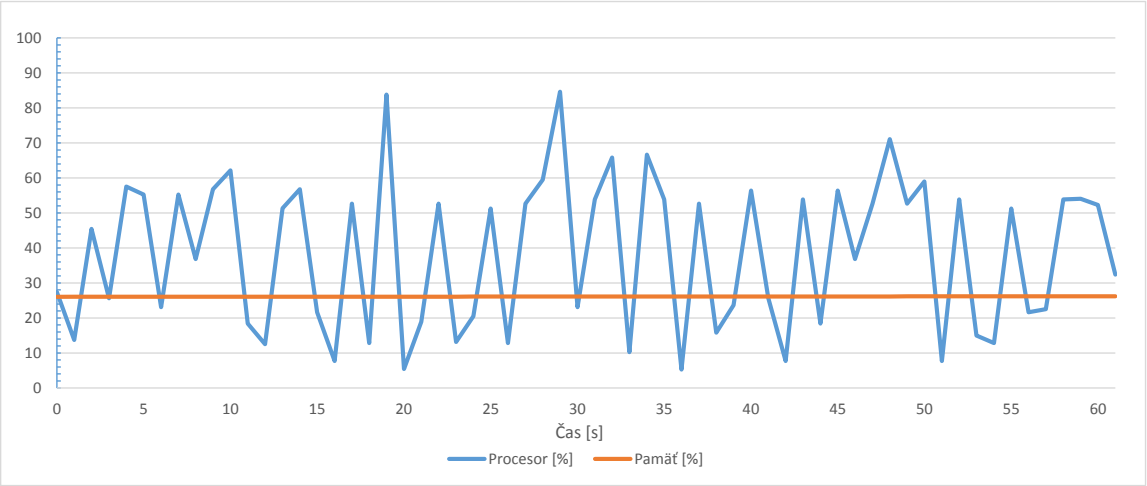
```
#!/usr/bin/env python3

'''Test for IXFR from Knot to NSD'''

from dnstest.test import Test

t = Test(tsig=False)

master = t.server("knot")
slave = t.server("nsd")
zones = t.zone_rnd(5, dnssec=False, records=50)

t.link(zones, master, slave, ixfr=True)

t.start()

# Wait for AXFR to slave server.
serials_init = master.zones_wait(zones)
slave.zones_wait(zones)

serials_prev = serials_init
for i~in range(4):
    # Update zone files on master.
    for zone in zones:
        master.update_zonefile(zone, random=True)
    master.reload()

    # Wait for IXFR to slave.
    serials = master.zones_wait(zones, serials_prev)
    slave.zones_wait(zones, serials_prev)
    serials_prev = serials

slave.flush()

t.end()
```

Obr. 6.13: Logovací výstup servera Knot

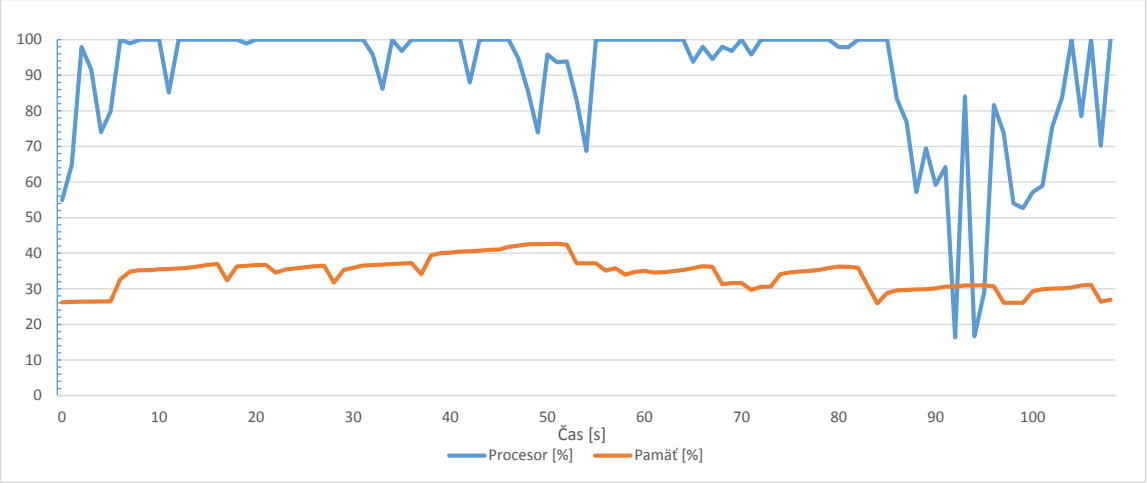
```
docker run -it cznic/knot:tests-extra parametre
```

Na testovanom počítači vykonanie všetkých testov sériovo trvalo približne jeden a pol hodiny. Pre demonštráciu paralelizácie boli použité množiny testov `basic` a `records`. Pri paralelnom testovaní boli použité dva kontajnery, každý z nich vykonával jednu sadu testov. Na obrázku 6.14 môžeme vidieť priebeh sériového testovania a sledované veličiny ako vyťaženie procesora a použitie systémovej pamäte. Obrázok 6.15 zobrazuje rovnaké veličiny avšak zobrazuje ich pri priebehu paralelného testovania. Meranie bolo vykonané pomocou programu `KSysGuard`, ktorý hodnoty veličín zapisoval každú sekundu do súboru.

Keďže testovací počítač obsahuje iba dvojjadrový procesor, nie je pri paralelizácii poznať žiadne zrýchlenie, dokonca sa pri paralelizácii čas vykonávania takmer zdvojnásobil. Môžeme si však všimnúť, že procesor bol oproti sériovému testovaniu viac vyťažený. Pamäť testovacieho počítača bola 4GiB. Zobrazená použitá pamäť obsahuje v sebe aj časť pamäte, ktorú použil operačný systém (∼750MiB). Sériové testovanie teda zobralo približne 300MiB, pričom paralelné približne 550MiB. Zvýšená spotreba operačnej pamäte je spôsobená virtualizáciou prostredia v kontajneroch a samotným softwarom bežiacim vnútri kontajnerov.



Obr. 6.14: Priebeh sériového testovania



Obr. 6.15: Priebeh paralelného testovania

Paralelizácia na výkonnejšom počítači by však priniesla značné výhody v podobe úspory času potrebného na testovanie. Čas paralelného vykonávania záleží hlavne na rýchlosti procesora. Vďaka paralelizácii je možné každý jeden test spustiť paralelne, preto najdlhšia teoretická doba testovania by mala byť rovnaká ako dĺžka vykonania najdlhšie trvajúceho testu.

Kapitola 7

Možnosti úprav frameworku v budúcnosti

Framework ponúka do budúcnosti veľa priestoru na zdokonalovanie a vylepšovanie. V nasledujúcom texte sú popísané niektoré body, ktoré by mohli byť v budúcnosti vylepšené. Jedná sa napríklad o časy behu testov, o spustenie testu s rovnakými dátami a nastaveniami a podobne.

Úpravu týkajúca sa flexibility pri zápise testov by bolo možné vylepšiť rozšírením funkčnosti jej základnej triedy `ConfigTree`. Programové rozhranie tejto triedy by bolo vhodné rozšíriť o metódy ktoré by ešte viac spríjemnili a zjednodušili jej používanie. Napríklad na jednoduchšiu lokalizáciu uzlu alebo atribútu v strome by bolo možné použiť takzvanú bodkovú notáciu.

V súčasnosti framework nepodporuje paralelizáciu. Na paralelizáciu testovania sa používa technika kontajnerov za použitia nástroja *Docker*. Táto paralelizácia teda prebieha na úrovni celého frameworku. Tak ako bolo spomenuté v kapitole 4.4.4, ktorá sa zaoberá požiadavkou na paralelizáciu frameworku, je táto paralelizácia vykonaná na najvyššej úrovni. V budúcnosti by bolo možné a vhodné upraviť framework tak, aby podporoval paralelizáciu aj bez použitia nástrojov tretích strán a aby bolo možné paralelizáciu vykonávať aj na úrovni celých testov alebo testovacích príkazov.

Implementácie DNS serverov použité v jednotlivých testoch sú spúšťané a zastavované frameworkom pri každom spustení alebo ukončení behu testu. Framework pri ich spúšťaní iba pasívne čaká určitú dobu a potom kontroluje, či server beží. Použitie týchto čakaní spolu s implicitnou serializáciou vykonávania testov predlžuje dobu sériového vykonávania testov. Čakanie je možné odstrániť buď úpravou implementácií pridaním systémových signálov na signalizáciu stavu alebo parsovaním výstupného logu serveru. Výhodnejšia by bola druhá, neinvazívna, varianta – parsovanie. Vyriešením týchto vlastností je možné skrátiť dobu vykonávania jednotlivých testov.

Pri vykonávaní niektorých testov má na výsledok vplyv aj časovanie akcií vykonaných vnútri frameworku, obsah náhodne vygenerovaných zón a prvky konfigurácie testu/frameworku, ktoré sú volené náhodne. Vďaka týmto faktorom môžu byť výsledky rovnakého testu pri každom vykonaní rôzne. Riešením by mohla byť možnosť znovuspustenia testu s rovnakými parametrami. Triedy frameworku by bolo možné serializovať a zónové súbory by

boli použité rovnaké ako v predchádzajúcom teste. Čím väčšiu množinu dát a nastavení by bolo možné znovu použiť, tým viac by klesla pravdepodobnosť, že test skončí s rozdielnym výsledkom.

Ďalšia vlastnosť, ktorá frameworku chýba, je dokumentácia zdrojového kódu. Nezainteresovaný užívateľ, ktorý framework začne používať a chce si vytvárať vlastné testy musí preštudovať zdrojový kód frameworku. Okrem zdrojového kódu neexistuje žiadna programová dokumentácia frameworku, ktorá by tento prvý krok pri používaní uľahčila.

Kapitola 8

Záver

Systém DNS je dôležitou súčasťou dnešného internetu. Jeho dostupnosť a správne fungovanie je teda kritické z pohľadu funkčnosti internetu. Združenie CZ.NIC vyvíja vlastný autoritatívny DNS server Knot DNS. Na jeho testovanie používa vlastný framework, ktorého úprava je predmetom tejto práce. Framework je spravovaný združením CZ.NIC spolu s implementáciu DNS servera Knot DNS. Vývoj frameworku teda bude v budúcnosti určite napredovať.

Súčasný upravený framework bol oproti pôvodnému frameworku obohatený o prvky ako *Dummy* server a *Blackbox*, upravený tak aby bolo čo najjednoduchšie pridať nové implementácie DNS serverov na testovanie a bola vytvorená dynamická dátová štruktúra spolu s obsluhujúcou triedou `ConfigTree` slúžiaca na uloženie a editáciu konfigurácie jednotlivých serverov. Za týmto účelom bolo vykonaných aj mnoho menších zmien, ktoré v práci neboli spomenuté.

Dummy server bol vytvorený ako viacvláknový TCP a UDP server, ktorý prijaté dáta poskytne užívateľovi na úpravu a následne ich odošle naspäť odosielateľovi. Užívateľ pritom môže prijaté dáta ľubovolne zmeniť.

Blackbox server (taktiež nazývaný *box-in-the-middle*) slúžiaci ako proxy server prepojuje dva DNS servery. Slúži na vytváranie umelých chýb v prenose, na vytváranie latencie a podobne. Toto všetko je v réžii užívateľa, ktorý chovanie definuje.

Pre podporu viacerých implementácií DNS serverov bola upravená časť frameworku a konkrétne triedy reprezentujúce DNS servery boli vyčlenené do samostatných modulov. Nové implementácie sú derivované zo základnej triedy a následne upravia potrebné funkcie pre zabezpečenie správneho chodu. Konfigurácie serverov sa v upravenom frameworku ukladajú do stromovej štruktúry.

Práce na úpravách frameworku boli vykonávané na lokálnej vetve verzovacieho systému Git a následne po úpravách na novú verziu serveru Knot DNS budú začlenené do hlavnej vetvy. V budúcnosti by sa dalo na frameworku vylepšiť a doplniť viacero vlastností. Niektoré z týchto vlastností boli popísané v kapitole 7.

Literatúra

[1] RFC 1034 - Domain names - concepts and facilities. `http://tools.ietf.org/html/rfc1034`, 1987-11 [cit. 2014-12-03].

[2] RFC 1035 - Domain names - implementation and specification. `http://tools.ietf.org/html/rfc1035`, 1987-11 [cit. 2014-12-03].

[3] A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY). `http://tools.ietf.org/html/rfc1996`, 1996-08 [cit. 2014-12-23].

[4] ALBITZ, P.; LIU, C.: *DNS and BIND*. Čtvrté vydání, 2001, ISBN 0-596-00158-4, xvii + 601 s.

[5] JORGENSEN, P. C.: *Software testing - A Craftsman's Approach*. CRC Press LLC, druhé vydání, 2002, ISBN 0-8493-0809-7.

[6] MATOUŠEK, P.: Predmet ISA – Systém DNS (kapitola 3), 2011/2012, zdroj FIT VUT v Brně.

[7] PATTON, R.: *Testování softwaru*. Computer Press, 2002, ISBN 80-7226-636-5.

[8] WWW stránky: Name Server Daemon (NSD) [online]. `http://www.nlnetlabs.nl/projects/nsd/documentation.html`, 2013-09-25 [cit. 2014-11-29].

[9] WWW stránky: BIND 9 Administrator Reference Manual [online]. `ftp://ftp.isc.org/isc/bind9/cur/9.10/doc/arm/Bv9ARM.pdf`, 2014 [cit. 2014-11-29].

[10] WWW stránky: Knot DNS 1.6.0 documentation [online]. `https://www.knot-dns.cz/static/documentation/singlehtml/`, 2014 [cit. 2014-11-29].

Dodatok A

Obsah CD

Priložený CD nosič obsahuje nasledovné položky:

- `doc`
  súbory so zdrojovými kódmi (v jazyku LaTeX) tejto písomnej práce a súbor `Makefile` na vytvorenie tejto práce
- `src`
  zdrojový kód frameworku
- `tests`
  výstupy testov vykonaných nad frameworkom
- `Framework na testování DNS serverů.pdf`
  PDF súbor obsahujúci túto prácu

Dodatok B

Manuál

Táto kapitola popisuje jednoduchý manuál pre užívateľa, ktorý chce začať používať tento testovací framework. Postupne stručne vysvetluje jednotlivé hlavné body, ktoré by mali užívateľovi pomôcť so základným testovaním. Pred začatím používania je vhodné aby si čitateľ prečítal prácu, ktorá objasňuje určité detaily frameworku a bude tak ľahšie pre čitateľa začať pracovať s frameworkom.

B.1 Minimálne systémové požiadavky

Beh samotného frameworku nie je náročný na systémové požiadavky, ide totižto iba o beh interpretu jazyka Python3. Minimálne systémové požiadavky teda závisia iba od testovaných implementácií DNS serverov. Samozrejmosťou je zohľadnenie počtu bežiacich DNS serverov v jeden moment. Toto všetko závisí od daného testu.

Implementácie DNS serverov NSD, Bind a Knot DNS majú približne rovnaké systémové požiadavky (čerpané z [8, 9, 10]):

- **hardwarové požiadavky** - požiadavky nie su príliš náročné, typická hardwarová zostava alebo virtualizačné riešenie by vo väčšine prípadov mali byť dostačujúce.
- **požiadavky na procesor** - dostačujúce by mali byť všetky dnešné procesory. Čím viac zón server musí obsluhovať, tým výkonnejší procesor potrebuje. S narastajúcim počtom výpočetného výkonu procesora narastá aj výpočetný výkon DNS servera.
- **pamäťové požiadavky** - pre server NSD je možné pamäťové požiadavky v závislosti od použitých zón vypočítať na adrese `http://www.nlnetlabs.nl/projects/nsd/nsd-memsize.html`, pre server Bind platí, že potrebuje také množstvo pamäte aby bola schopná uložiť cache a zónové súbory načítané z disku a pre server Knot DNS sa odhaduje spotreba pamäte na trojnásobok veľkosti zónových súborov v textovom formáte.
- **požiadavky na operačný systém** - všetky uvedené DNS servery bežia na väčšine unixových systémov.

B.2 Prerekvizity

Potrebné prerekvizity pre samotný framework sú uvedené v súbore `./README`. Sú to nasledovné:

- Python verzie minimálne 3.3
- python3-dnspython verzie minimálne 1.11.1
- psutil (sudo pip3 install psutil)
- dnssec-signzone
- dnssec-keygen
- dnssec-verify
- Bind 9.9.x
- NSD 3.2.x
- lsof
- gawk
- (valgrind)
- (gdb)

B.3 Inštalácia

Všetky uvedené prerekvizity si musí užívateľ nainštalovať sám. Postup inštalácie niektorých súčastí a iné dôležité nastavenia pri inštalácii sú uvedené v súbore `./README`. Na inštaláciu Python súčastí možno použiť príkaz „`pip install -r requirements.txt`“. Pri inštalácii na operačnom systéme Ubuntu musí užívateľ vykonať zmeny v systéme uvedené v súbore `./README` v sekcii „Ubuntu“.

B.4 Použitie frameworku

Pre spustenie testovania slúži skript `./runtests.py`. Skript berie ako parametre zoznam testov, ktoré majú byť vykonané, poprípade zoznam testov, ktoré majú byť vynechané. Na základe predaných parametrov začne spúšťať jednotlivé testovacie skripty a zaznamenáva ich priebeh vykonávania. Synopsia skriptu je nasledovná:

```
runtests.py [-h] [-d] [[:]test[/case] [[:]test[/case] ...]]
```

Pozičné parametre:

```
[:]test[/case] ([exclude] | run) specific (test set | [test case])
```

V nasledújúcich podkapitolách sú uvedené základné body týkajúce sa používania frameworku. Všetky detaily použitia frameworku, ktoré nie sú uvedené v tomto manuále musí užívateľ nájsť priamo v zdrojovom kóde frameworku.

B.4.1 Zónové súbory

V testoch možno použiť dopredu pripravené zónové súbory alebo vygenerovať náhodné. V prvom prípade možno pripravené zónové súbory uložiť na dve umiestnenia. Zónový súbor je vždy umiestnený v priečinku `./data`, rozdiel je v tom, že pri prvom umiestnení je priečinok v koreňovom adresáre frameworku a pri druhom umiestnení je priečinok v adresáre testu. Na vytvorenie objektu zóny, ktorá sa následne použije v teste sa používa nasledujúceho príkazu:

```
zone = t.zone(ëxample.", storage=".")
```

Príkaz vytvorí objekt triedy `ZoneFile`. Načítaná bude zóna zo súboru `example.zone` z priečinku `./data` umiestneného v priečinku testu. Pokial nie je parameter `storage` uvedený, použije sa priečinok `data` z koreňového adresára frameworku.

B.4.2 Vytvorenie vlastného testu

Pre vytvorenie vlastného testu treba vytvoriť priečinky na ceste k novému testu (umiestnenie uvedené relatívne ku koreňovému adresáru frameworku):

```
./tests/NAZOV-SADY-TESTOV/NAZOV-TESTU/
```

Do priečinka nového testu následne vložíme testovací skript s pevne daným názvom `test.py`. Každý testovací skript sa približne drží nasledujúcej schémy implementácie:

1. vytvorenie objektu triedy Test
2. získanie objektov reprezentujúcich jednotlivé implementácie DNS serverov z vytvoreného objektu triedy Test
3. konfigurácia serverov
4. (vygenerovanie a) nastavenie zónových súborov serverov
5. spustenie testu pomocou objektu triedy Test
6. konkrétne príkazy špecifické pre testovanie jednotlivých vlastností serverov
7. ukončenie testu pomocou objektu triedy Test

Príklad konkrétneho kódu testu je uvedený na obrázku B.1.

Metódy, ktoré podporujú rozhrania tried `Test`, `Server` a iné je možné vyčítať z kódu frameworku.

```
#!/usr/bin/env python3

from dnstest.utils import *
from dnstest.test import Test

t = Test()
knot = t.server("knot")
zone = t.zone("flags.")

t.link(zone, knot)

t.start()
...
t.end
```

Obr. B.1: Príklad základnej kostry testu

B.4.3 Výsledky testov

Každé spustenie frameworku vytvorí vo vybranom priečinku adresárovú štruktúru, do ktorej sú následne uložené výsledky priebehu testov. Umiestnenie tohto priečinku je možné zmeniť priamo v skripte `./tools/dnstest/params.py` v premennej `outs_dir` alebo nastavením premennej prostredia `KNOT_TEST_OUTS_DIR` na požadované umiestnenie.

Adresárová štruktúra na obrázku B.2 bola vytvorená s prednastaveným umiestnením v priečinku `/tmp/`. Priečinok `knottest-`*vygenerovana-sekvencia-1-az-N* reprezentuje jeden beh frameworku. V priečinku sa nachádzajú podpriečinky obsahujúce jednotlivé množiny testov. Na ďalšej úrovni sú to už výsledky konkrétnych testov a iné dáta použité pri testoch (konfiguračné súbory serverov, zónové súbory, chybový/štandardný výstup serverov, kľúče a iné). Súbor `case.log` obsahuje logovací výstup konkrétneho testu a `traceback.log` obsahuje chybový výstup interpreta jazyka Python 3.

```
tmp/ ..... koreňový priečinok výsledkov
└── knottest-vygenerovana-sekvencia-1-az-N/
    ├── nazov-skupiny-testov-1-az-N/ ..... skupina určitých testov
    │   └── nazov-testu-1-az-N/ ..... výsledky konkrétnych testov
    │       ├── nazov-serveru-1-az-N/ ..... dáta jednotlivých DNS serverov
    │       ├── zones/ ..... zónové súbory použité v teste
    │       ├── case.log ..... logovací výstup testu
    │       └── traceback.log ..... výstup testu pri chybe
    ├── environment.log ..... zaznamenané premenné prostredia
    ├── failed.log ..... zoznam neúspešných testov
    └── summary.log ..... výsledný sumár testov
└── knottest-last ..... symlink na priečinok s výsledkami posledného testu
```

Obr. B.2: Štruktúra adresára výsledkov testov

Priečinok `knottest-`*vygenerovana-sekvencia-1-az-N* obsahuje ešte tri logovacie súbory. Do súboru `environment.log` sa zaznamenávajú hodnoty premenných prostredia použitých pri behu, súbor `failed.log` obsahuje zoznam neúspešných testov s jednoduchým popisom chyby a posledný súbor `summary.log` obsahuje jednoduchý zápis behu frameworku spolu s časovými razítkami jednotlivých udalostí.

V prípade testov, kedy je server dotazovaný na záznamy priamo z rozhrania frameworku (metóda `Server.dig()`), sú jednotlivé dotazy zaznamenávané do súboru `case.log`. Tento súbor teda v prípade chyby obsahuje záznam, pri ktorom dotaze k chybe došlo.

Symbolický odkaz `knottest-last` slúži na prístup do priečinku obsahujúceho výsledky posledného vykonaného behu frameworku.

B.4.4 Pridanie podpory pre ďalšiu implementáciu DNS serveru

Pre pridanie podpory novej implementácie DNS servera je potrebné vytvoriť v priečinku `./tools/dnstest/server/` nový modul s názvom súboru, ktorý sa bude zhodovať s názvom triedy reprezentujúcej server, pričom sa neberie do úvahy veľkosť písmen. Napríklad teda v module `exampleserver.py` bude za triedu reprezentujúcu server určená trieda s menom ako napríklad „ExampleServer“, „exampleServer“, a pod.

Trieda vytvorená v module musí byť derivovaná z abstraktnej (základnej) triedy `BaseServer` umiestnenej v priečinku `./tools/dnstest/`. V triede musia byť implementované minimálne metódy `get_config` a `listening`. Metóda `get_config` slúži na vygenerovanie textovej reprezentácie konfiguračných súborov pre dané implementácie DNS serverov a metóda `listening` sa používa na zistenie pripravenosti servera.

Každá nová implementácia DNS servera musí v module `params.py` definovať cestu k binárnemu súboru, ktorým sa DNS server spúšťa a cestu k binárnemu súboru, ktorý ovláda daný server. Na tento účel slúži funkcia `get_binary`, ktorá vyhľadá systémovú premennú zadaného názvu - prvý argument funkcie. Ak táto premená v systéme nie je definovaná, ako cesta k binárnemu súboru je použitá cesta uvedená ako druhý argument funkcie. Príklad zadania ciest k binárnym súborom serveru NSD je na obrázku B.3.

```
1 # KNOT_TEST_NSD - Nsd binary.
2 nsd_bin = get_binary("KNOT_TEST_NSD", "nsd")
3 # KNOT_TEST_NSDC - Nsd control binary.
4 nsd_ctl = get_binary("KNOT_TEST_NSDC", "nsdc")
```

Obr. B.3: Príkazy definujúce binárne súbory DNS serveru NSD

Následne je potreba implementovať všetky metódy základnej triedy, ktoré nevyhovujú pre účely testovania novej implementácie DNS servera.

B.4.5 Použitie *Dummy* serveru

V niektorých prípadoch testovania je potreba posielať predpripravené odpovede testovaným serverom. `Dummy` server slúži ako jednoduchý TCP/UDP server, ktorý prijíma dáta na svojom porte a vo forme bitového poľa (typ `bytearray`) ich posiela do užívateľom definovanej funkcie. Dáta vrátené z užívateľskej funkcie sú následne zaslané späť odosielateľovi.

Spôsob použitia je demonštrovaný na obrázku B.4. Server sa používa skoro rovnako ako iné servery, jedinými odlišnosťami sú:

- vytvorenie užívateľskej funkcie a jej priradenie do premennej serveru,

- zapnutie/vypnutie logovania prichádzajúcich dát pomocou príznaku `log_traffic`.

```
#!/usr/bin/env python3
''' Príklad použitia Dummy serveru. '''

from dnstest.utils import *
from dnstest.test import Test

# užívateľská funkcia
# parametre: dáta, reťazec reprezentujúci transportný protokol ("tcp"/"udp")
def my_handler(data, protocol):
    # spracovanie dát a~ich vrátenie (typ byteArray)
    return data

t = Test(tsig=False)
dummy = t.server("dummy")       # vytvorenie objektu Dummy serveru
dummy.DIG_TIMEOUT = 5           # hodnota timeoutu pre odpoveď serveru
dummy.user_handler = my_handler # užívateľská funkcia ktorá bude použitá

t.start()
# príklad dotazu na Dummy server
dummy1.dig("domainname", "A", udp=True, tries=1)
t.end()
```

Obr. B.4: Príklad použitia *Dummy* serveru

B.4.6 Použitie prvku *Blackbox*

Blackbox slúži ako proxy server na spojení medzi dvoma DNS servermi a ich komunikácia prebieha prostredníctvom tohto prvku. Užívateľ má možnosť pristupovať k tejto komunikácii a ľubovolne s ňou manipulovať. To znamená, že dáta môže ľubovolne upraviť, vymazať, oneskoriť a podobne. *Blackbox* teda slúži na simulovanie chýb pri prenose ako napríklad vytváranie chýb na úrovni bitov, vytváranie latencie linky medzi servermi a podobne.

Použitie *Blackbox* je demonštrované na obrázku B.5. Vytvorenie je rovnaké ako u iných serverov. *Blackbox* podporuje zopár nastavení, ktoré sú pre neho špecifické. Nastavenie logovania prichádzajúcich dát je nastavené rovnako ako v prípade *Dummy* serveru pomocou príznaku `log_traffic`. Ďalej musí užívateľ definovať dve premenné:

- server1_ingress_user_handler - obsahuje adresu funkcie, ktorej sa predajú dáta pred vstupom do master (podľa funkcie `link`) servera,
- server2_ingress_user_handler - obsahuje adresu funkcie, ktorej sa predajú dáta pred vstupom do slave (podľa funkcie `link`) servera.

Pre správne fungovanie musí byť *blackbox* vložený „medzi“ dva servery. To je zabezpečené funkciou `link`, ktorá ako pomenovaný parameter `blackbox` prijíma objekt triedy `Blackbox`. Následne je *blackbox* pripravený na použitie.

B.4.7 Paralelné testovanie

Paralelizáciu je možné vykonať pomocou nástroja `Docker`, ktorý používa techniku kontajnerov. Obraz pre vytvorenie kontajnera je možné vytvoriť pomocou nástroja `docker` a súboru `Dockerfile` alebo jednoducho spustiť daný kontajner a nástroj si najnovší obraz,

```
#!/usr/bin/env python3
''' Príklad použitia Blackboxu. '''

from dnstest.test import Test
from dnstest.utils import *

# užívateľská funkcia
# parametre: dáta, cieľový server, reťazec reprezentujúci transportný protokol ("tcp"/"udp")
def my_handler(data, to_server, protocol):
    return data

t = Test(tsig=False)
master = t.server("bind")
slave = t.server("knot")

# Vytvorenie Blackboxu
blackbox = t.server("blackbox")
# definovanie užívateľských funkcií
blackbox.server1_ingress_user_handler = my_handler
blackbox.server2_ingress_user_handler = my_handler

# vytvorenie a~prepojenie serverov na základe danej zóny
zones = t.zone("flags.")
t.link(zones, master, slave, blackbox=blackbox)

t.start()
...
t.end()
```

Obr. B.5: Príklad použitia *Blackbox*

ktorý bude spustený v kontajneri stiahne online z *Docker Hub* úložiska. Všetky potrebné inštrukcie sú uvedené v súbore `README`.

Aktuálny obraz `cznic/knot:tests-extra`, ktorý je dostupný na *Docker Hub* úložisku je možné stiahnuť a zároveň spustiť po nainštalovaní *Docker* nástroja pomocou príkazu:

```
docker run -it cznic/knot:tests-extra parametre
```

Na vytvorenie vlastného obrazu je možné použiť nasledujúci príkaz, ktorý z aktuálneho umiestnenia nakopíruje do obrazu testovacie skripty:

```
docker build -t tests-extra .
```