Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Dynamická konfigurace a správa modulárního DNS resolveru

Diplomová práce

Bc. Aleš Mrázek

Školitel: Ing. Ladislav Lhotka, CSc.

Garant: Ing. Jan Fesl, Ph.D.

Konzultant: Ing. Petr Špaček

České Budějovice 2020

Bibliografické údaje

Mrázek, A., 2020: Dynamická konfigurace a správa modulárního DNS resolveru. [Dynamic configuration and administration of DNS resolver. Mgr. Thesis, in Czech.] – 106 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Diplomová práce se zabývá návrhem systému určeného k lokální i vzdálené administraci open-source implementace DNS resolveru Knot Resolver, která se vyznačuje neobvykle vysokou modularitou a je realizován několika samostatnými procesy konfigurované jazykem Lua. Práce uvádí problémy spojené s administrací Knot Resolveru a navrhuje řešení v rámci systému založeného na NETCONF protokolu. Pro definici konfigurace a administračních operací je vytvořen datový model pomocí jazyka YANG. K návrhu a realizaci systému je využita sada nástrojů libyang, sysrepo a netopeer2, které jsou založené na na otevřených standardech NETCONF protokolu a modelovacího jazyka YANG.

Klíčová slova

DNS, Knot Resolver, NETCONF, YANG, sysrepo, konfigurace, administrace

Annotation

The master thesis deals with the design of a system for local and remote administration of the open-source DNS resolver Knot Resolver, which is characterized by unusually high modularity and is implemented by several separate independent processes configured by Lua language. The thesis presents the problems associated with the administration of Knot Resolver and proposes a solution within the system based on the NETCONF protocol. A data model is created using the YANG language to define configuration and administrative operations. Toolsets libyang, sysrepo and netopeer2, based on the open standards of the NETCONF protocol and the YANG modeling language, are used to design and implement the system.

Keywords

DNS, Knot Resolver, NETCONF, YANG, sysrepo, configuration, administration

Prohlašuji, že svoji diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne ..... Podpis autora .....

Poděkování

Rád bych touto cestou poděkoval Ing. Ladislavu Lhotkovi, CSc. a vývojovému týmu Knot Resolveru pod vedením Ing. Petra Špačka za možnost pracovat na tomto tématu a za jejich cenné rady a spolupráci. Mé díky také patří vývojářům pracujícím na sadě nástrojů libyang, sysrepo a netopeer2 za jejich rady a pomoc. Nakonec bych rád poděkoval Ing. Janu Feslovi, Ph.D. za ochotu být garantem této práce.

Obsah

Úvod

Počítačové sítě a síťové technologie se neustále rozvíjejí a rozšiřují. Kondice takové sítě nebo služeb, jako je i Domain Name System (DNS), závisí na správné konfiguraci prvků v této síti. S narůstající velikostí sítě je tyto prvky stále obtížnější efektivně konfigurovat, řídit nebo monitorovat. Jistě tomu nepomáhá ani fakt, že jednotlivá zařízení a služby často poskytují naprosto odlišné rozhraní a způsoby pro administraci, to platí i u jednotlivých implementací DNS resolveru, i když se jejich konfigurace principiálně neliší a provádějí velmi podobné operace. Existují nástroje, například Ansible[1], Chef[2] nebo Puppet[3], umožňující částečnou automatizaci. Tyto nástroje je ale potřeba na každém jednotlivém zařízení manuálně nastavit, jelikož mezi konfiguračními rozhraními jednotlivých zařízení není dohodnutý jakýkoliv společný způsob konfigurace. Administrace sítě se tak stává časově náročnou a neefektivní, s čímž zároveň rostou i finanční a lidské zdroje potřebné k provozu sítě. Proto byl organizací IETF (Internet Engineering Task Force) navržen nový protokol zvaný NETCONF, který standardizuje dohodnutý mechanizmus pro síťovou konfiguraci.

Knot Resolver[4], vyvíjený v laboratořích sdružení CZ.NIC, je jednou z několika široce používaných open-source implementací DNS resolveru. Od ostatních implementací se odlišuje svou speciální architekturou, která je výrazně modulární. Vyznačuje se tím, že Knot Resolver je systém reprezentovaný několika na sobě nezávislými procesy a samostatnými moduly, které je možné vytvářet a upravovat jimi celkovou funkcionalitu. Další zajímavostí je způsob jeho konfigurace, která je realizovaná pomocí skriptu v jazyce Lua. Tato architektura, ačkoli má mnoho výhod, zásadním způsobem komplikuje celkovou administraci. Jednotlivé procesy je z důvodu absence centrálního prvku a komunikace mezi procesy potřeba řídit manuálně, konfigurovat a monitorovat samostatně. Dále použití Lua jazyka ke konfiguraci je pro administrátory, kteří nejsou programátoři, zbytečně komplikované a nepřehledné.

[1] `https://www.ansible.com/`
[2] `https://www.chef.io`
[3] `https://www.puppet.com`
[4] `www.knot-resolver.cz`

Podobně jako může být neefektivní síťová konfigurace zařízení v síti zlepšena použitím NETCONF protokolu, je i v případě Knot Resolveru možné použít tento protokol. Serverová část NETCONF protokolu specifikuje datová úložiště (datastores) pro správu konfigurace, ze které mohou jednotlivé procesy konfiguraci odebírat. V konfiguračním datovém modelu, který je definován jazykem YANG a určuje strukturu dat uložených v datovém úložišti, je také možné definovat operace nebo notifikace, které mohou sloužit k řízení nebo monitorování. Jako implementace NETCONF protokolu jsou v práci využity nástroje sysrepo[5] a netopeer2 vyvíjené sdružením CESNET. Pro návrh datového modelu je jako báze použit průnik konfigurací několika dalších open-source implementací DNS resolveru, která je rozšířena o datový model specifický pro Knot Resolver. Provozovatelé sítí totiž velmi často používají několik implementací DNS resolveru zároveň. Společná část konfigurace by jim umožnila snadnou přenositelnost konfigurace nebo možnost konfigurovat více rozdílných implementací najednou. Ovšem za předpokladu, že i jiné implementace DNS resolveru využijí tuto společnou část konfigurace.

Tato práce si klade za cíl navrhnout systém s využitím NETCONF protokolu a modelovacího jazyka YANG, který zlepší v současnosti problematickou administraci Knot Resolveru zapříčiněnou jeho modularitou a konfigurací jazykem Lua. Navržený systém by měl jednotlivé procesy ovládat, konfigurovat a monitorovat centrálně z jediného konfiguračního rozhraní a to i vzdáleně.

[5] `https://www.sysrepo.org`

Metodika práce

Diplomová práce je zaměřena na návrh prototypu systému založeného na NETCONF protokolu, který bude relativně samostatnou vrstvou pro administraci DNS resolveru Knot Resolver, který je realizovaný několika samostatnými procesy. Jako výchozí postup pro řešení budou provedeny dvě metody výzkumu, tou první je literární rešerše zkoumaného tématu a tou druhou je dotazování neboli zjišťování požadavků na budoucí systém. Výstupy z těchto postupů budou vstupními parametry pro prvotní návrh prototypu systému.

Problematika řešena v rámci práce zasahuje do několika různých odvětví informačních technologií, díky tomu je rozsah tématu relativně široký. Proto nejdříve bude nastudována jak obecná rovina problému, tak i konkrétní technologie. Ta nejdůležitější témata k prostudování jsou DNS, Knot Resolver, NETCONF protokol a sysrepo. Než bude možné začít se samotným návrhem systému bude potřeba probrat detailní požadavky na jeho funkce a očekávání s vývojovým týmem Knot Resolveru. K tomuto účelu bude využit konzultant práce, který je vedoucím projektu Knot Resolver, má potřebné informace nebo ví, kde a jak je získat.

Na základě získaných informací a požadavků z předchozích postupů bude navrženo řešení pro jednotlivé problémy, které se společně s vybranými technologiemi spojí v novém návrhu systému. Po ujasnění teoretického návrhu bude přikročeno k realizaci jednotlivých částí nového systému. K vývoji bude přistupováno na způsob prototypového přístupu, kdy budou postupně iterativně implementovány jednotlivé funkce a jednotlivé kroky nebo případné problémy budou pravidelně konzultovány s vedoucím a konzultantem práce.

1 Domain Name System

Domain Name System (DNS) nepochybně spadá mezi nejstarší a nejdůležitější služby sítě Internet, přesto se pro většinu internetových uživatelů jedná o službu takřka neviditelnou. Již od počátků používají počítačové sítě pro identifikaci zařízení v síti číselné IP adresy, které jsou však pro člověka nepohodlné a obtížně zapamatovatelné. Proto se v minulosti začaly vymýšlet mechanismy jak uživatelům umožnit pojmenovat zařízení připojené do sítě a následně tato jména používat namísto číselných adres. Hlavním úkolem těchto mechanismů byl vzájemný převod mezi jmény a číselnými adresami. DNS je v současnosti nejrozšířenějším mechanismem, který toto umožňuje jak v sítích internetových, tak i privátních. Jedná se o rozsáhlý hierarchický systém realizovaný DNS servery na jehož provozu, údržbě a zlepšování se podílí nespočet lidí a organizací z celého světa.

1.1 Historie

Historický kontext vzniku sahá až do šedesátých let minulého století, kdy jedna z agentur ministerstva obrany Spojených států amerických *ARPA (The Advanced Research Projects Agency)*, později přejmenována na DARPA (The Defense Advanced Research Projects Agency), začala financovat vývoj počítačové sítě ARPANET. Počty připojených počítačů velmi rychle rostly k deseti tisícům a tak vznikl předchůdce globální sítě dnes velmi dobře známé jako Internet. [3, 41]

Již v počátcích ARPANETu se s rostoucím počtem počítačů objevovaly snahy o vytvoření centrálního seznamu počítačů a jeho použití k pojmenování počítačů jednoduše zapamatovatelnými jmény namísto číselných adres. Následně vznikl způsob nazvaný jako *Hostname Server* popsaný v RFC 811 [27] a později RFC 953 [26]. Jednalo se o způsob s jedinou centrální databází, na kterou probíhalo dotazování na konkrétní jména až v okamžiku, kdy jej klient potřeboval. Poskytuje tedy aktuální informace, ale s rostoucím počtem klientů naráží na problém své škálovatelnosti. V dimenzích současného Internetu by takový systém převodu jmen na adresy nemohl v žádném případě fungovat. [41]

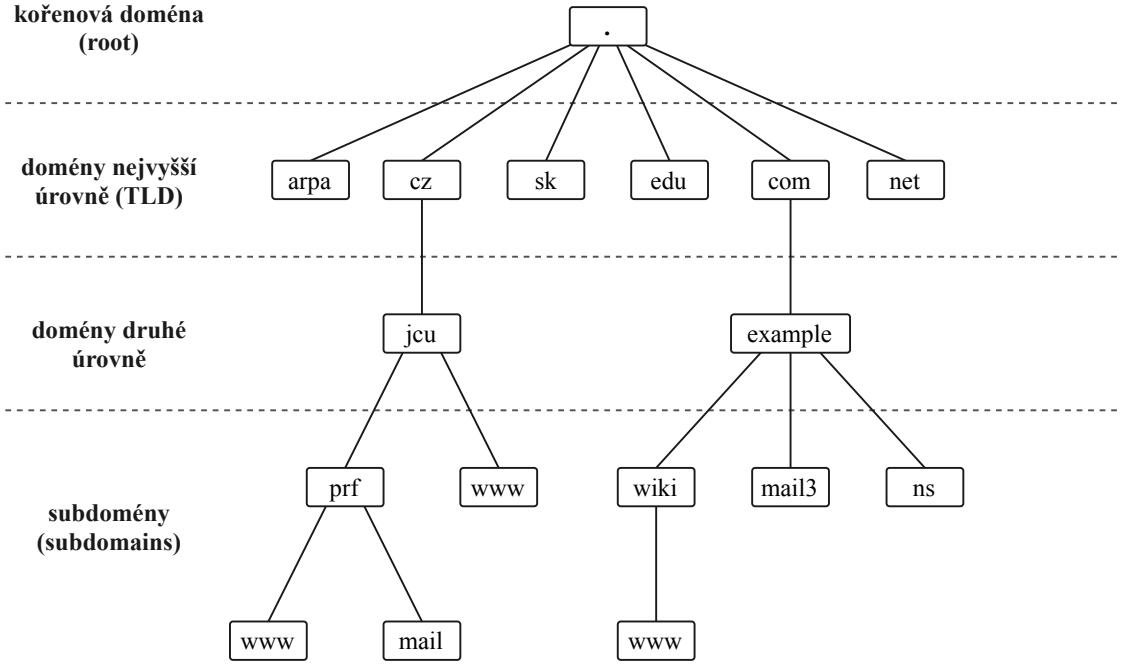
Díky znalostem a zkušenostem získaných z předchozích mechanizmů a jejich nedostatků bylo DNS vytvořeno jako silně distribuovaný a redundantní systém. Při jeho návrhu sehrály zásadní roli následující myšlenky. [3]

- Konzistentní systém jmen identifikující jednotlivé zdroje.
- Poskytovat stále aktuální údaje.
- Distribuovaná správa dat, tak aby informace zadával pokud možno místní správce.
- Systém musí být schopen reagovat na rychle se měnící údaje.
- Systém by měl být efektivní a co nejméně zatěžovat komunikační infrastrukturu.
- Dostupnost údajů je důležitější než jejich konzistence.

První plná implementace DNS se nazývala *JEEVES*. Autorem je Paul Mockapetris, který zároveň sepsal základní specifikaci DNS publikované roku 1983. Následující rok vytvořil Kevin Dunlap implementaci pro operační systém *Berkley 4.3 BSD Unix* nazvanou jako *Berkeley Internet Name Domain*, dnes velmi dobře známou jako pod zkratkou *BIND*. Tato implementace se za krátkou dobu prosadila takřka do všech variant Unixu a z něj odvozených systémů a dodnes jej používá většina serverů. V současnosti je BIND pod správou společnosti *ISC (Internet System Consortium)*. Zkušenosti s provozem prvních DNS se promítly do druhé generace jeho specifikace. V roce 1987 vyšla nová specifikace dokumentů RFC 1034 [44] a RFC 1035 [45], jejichž autorem je opět Paul Mockapetris a která nahradila původní specifikaci. Tyto dva dokumenty popisující základní funkce DNS zůstávají v platnosti dodnes, ačkoli existuje pro DNS i několik následných změn a rozšíření. Mezi významná rozšíření patří například *Domain Name Security Extensions* (*DNSSEC*, RFC 4035 [5]) umožňující ověřit pravost DNS záznamu a *DNS64* (RFC 6147 [7]) umožňující klientovi z IPv6 sítě přístup k službám sítě provozující pouze protokol IPv4. [3, 41]

1.2 Domain Namespace

V českém překladu se často používá slovní spojení *doménový strom*. Jak už české znění napovídá jedná se o stromovou datovou strukturu s jedním kořenem, která slouží k hierarchickému uspořádání do ní uložených dat, v případě DNS jsou to doménová jména. Toto uspořádání, které je specifikováno v RFC 882 [43] vizualizuje následující obrázek 1.1.



Obrázek 1.1: Ukázka hierarchie doménového stromu

1.2.1 Doménová jména a domény

Doménová jména (Domain Names) jsou jednoznačná označení jednotlivých uzlů v doménovém stromě (DomainNamespace). Každý jeho uzel má svou textovou reprezentaci, která může obsahovat až 63 bajtů. Doménové jméno konkrétního uzlu je tvořeno posloupností textových reprezentací navštívených uzlů směrem ke kořenu oddělených tečkou. Pro kořenový uzel je vyhrazen řetězec o nulové délce, tedy prázdný řetězec. Z tohoto důvodu zůstává na konci doménového jména samostatná tečka `www.prf.jcu.cz.`, kterou je možné při zápisu doménového jména vynechat. Zápis doménového jména s tečkou na konci je tzv. plně kvalifikované doménové jméno (FQDN, Fully Qualified Domain Name).

Jména bez koncové tečky jsou někdy interpretována jako relativní k jinému uzlu než je kořenový, podobně jako je to s úvodním lomítkem u adresářů v Unixu/Linuxu. [41]

Domény (Domains) jsou jednoduše řečeno podstromy doménového stromu. Domény a jednotlivé uzly jsou reprezentovány doménovými jmény (Domain Names), přičemž jméno domény a nejvyššího uzlu té samé domény jsou stejné. Například pro doménu `jcu.cz` je nejvyšším uzlem `jcu.cz`. Doména může mít několik dalších podstromů nazývané jako subdomény (subdomains), které končí doménovým jménem nadřazené domény. Například `prf.jcu.cz` je subdoménou `jcu.cz`. Domény se dělí do několika kategorií podle jejich pozice v doménovém stromě. [41]

Kořenová doména (root) Jedná se o jedinou doménu na této úrovni, proto nemá smysl ji neustále zapisovat a standardně se nezapisuje. V případě, že je nutné kořenovou doménu reprezentovat, používá se pro její zápis samostatná tečka, kterou je možné vidět i v ukázce doménového stromu na obrázku 1.1. [41]

Domény nejvyšší úrovně (TLD) Pro domény nejvyšší úrovně se používá zkratka TLD z anglického Top-Level Domain. Tyto domény jsou o úroveň níže než je kořenová doména a tvoří základní členění domén. V raných počátcích Internetu bylo jejich původním cílem jasně rozlišit uživatele domény, v té době existovalo pouze šest domén této úrovně, `com, org, net, int, edu, gov` a `mil`.

Průběžně s expanzí Internetu toto základní rozdělení přestávalo být schopné pokrýt poptávku celého světa, a tak se začaly domény nejvyšší úrovně rozšiřovat. V současné době existuje pod správou *IANA (Internet Assigned Numbers Authority)* více než 1500 TLD [32] rozdělených do několika základních kategorií. [3, 41]

- **Generické domény** (generic top-level domains, gTLD):
  Do této kategorie patří právě většina domén z období raného vzniku DNS doplněné o některé další domény jako je například `info`.

- **Omezené generické domény** (generic-restricted top-level domains):
  Tyto domény patří také mezi generické, ale majíc přívlastek omezené (restricted). Je možné je registrovat pouze za určitých podmínek pro specifickou doménu. Mezi tyto domény patří `biz, name, pro`.

- **Sponzorované domény** (sponsored top-level domains):
  Tyto domény jsou také omezeny. Každá doména má svého sponsora, který určuje pravidla pro registraci subdomén. Patří sem `edu, gov, int, mil` a dále například `jobs, museum` či `asia`.

- **Státní domény** (country-code top-level domains, ccTLD):
  Jedná se o domény přidělené jednotlivým státům. Odpovídají až na výjimky dvoupísmenným zkratkám států podle standardu ISO 3166-1. Mezi výjimky patří například `eu` nebo Velká Británie `uk`.

- **Infrastrukturní doména** (infrastructure top-level domains):
  Jde pouze o jednu jedinou doménu s názvem `arpa`, která slouží pro interní mechanismy Internetu, například při původním zavádění DNS v ARPANET. Pojmenována je historicky po agentuře *ARPA*. Zkratka ovšem nově znamená *Adress and Routing Parameter Area*.

Domény nižších úrovní Hloubka doménového stromu je omezena na 127 úrovní. V praxi se však zřídka setkáme s hloubkou větší než je pět. Subdomény domén nejvyšší úrovně jsou domény tzv. druhé úrovně, ty bývají spravovány dvěma základními způsoby.

- Správce TLD přímo přiděluje subdomény konkrétním organizacím či jednotlivcům. Tímto způsobem je spravována i česká doména `cz`.

- Správce TLD se nejprve snaží pomocí domény druhé úrovně rozlišit charakter vlastníka podobně jako v původních doménách `com` nebo `edu` a přiděluje tedy až doménu třetí úrovně. Nejznámějšími zástupci tohoto přístupu jsou Rakousko, Velká Británie či Austrálie. Tento způsob je ojedinělý a spíše se od něj ustupuje. Například v Rakousku je již možné si zaregistrovat doménu druhé úrovně. [41]

1.3 Jmenné servery a zóny

Jmenné servery (Nameservers) jsou serverové programy, které obsahují informace o konkrétním uzlu doménového stromu. Jmenné servery zpravidla poskytují kompletní informace o některé části doménového stromu (Domain Namespace), která se nazývá *zóna*. Takovéto servery jsou pak nazývány jako autoritativní pro danou zónu. DNS specifikace definuje dva druhy jmenných serverů, primární a sekundární. Oba tyto servery jsou autoritativní, rozdíl je v získání zónových dat. V ideálním případě by měl totiž administrátor upravovat zónová data pouze v primárním jmenném serveru, od kterého si sekundární servery, kterých bývá z pravidla více, stáhnou data automaticky. Tomuto mechanizmu se říká zone transfer. Rozdíl mezi zónou a doménou je důležitý, i když ne nějak obrovský. Veškeré domény nejvyšší úrovně (TLD), mnoho domén na druhé a nižší úrovni jsou delegovány na menší, administrátorsky lépe zvládnutelné části neboli zóny. [41, 3, 44]

Jmenné servery kořenové zóny Jedná se o síť stovek serverů rozmístěných po celém světě, které jsou společně nakonfigurovány jako 13 autoritativních jmenných serverů. [31] Aby bylo DNS funkční musí být informace o těchto serverech veřejně dostupné. [3] Představují zásadní část technické infrastruktury Internetu, jelikož poskytují informace o TLD jmenných serverech důležité pro rekurzivní dotazování, viz 1.6. [41]

1.4 Resolvery

Programy, které na základě požadavků od klienta získají požadovaný záznam z jmenných serverů se nazývají DNS resolvery. Chovají se jako prostředník mezi klientem a jmennými servery, přičemž resolver může s dotazem od klienta zacházet několika způsoby. [44]

- **Rekurzivní resolver** Ve většině případů mají resolvery vlastní cache paměť do které si ukládají již vyřešené dotazy aby nemusely opakovaně vytvářet ty samé dotazy na jmenné servery. Pokud je již požadovaná informace klientem v cache paměti (cache hit), DNS resolver může rovnou odpovědět a tím ušetřit mnoho času a dalších zdrojů. Pokud informace v cache paměti není (cache miss) musí se resolver začít ptát jmenných serverů takzvanými rekurzivními dotazy (kapitola 1.6). Resolver, který provádí rekurzivní dotazovaní se nazývá Recursive resolver nebo také DNS recursor.

- **Přeposílající resolver** Tento DNS resolver neprovádí rekurzivní dotazování, ale pouze přeposílá (forwarding) dotazy na rekurzivní resolver, který pro něj získává odpověď. Tomuto DNS resolveru se říká Forwarding resolver nebo také DNS forwarder.

- **Stub resolver** Jedná se o jednoduchý program zabudovaný často v operačním systému, který ostatním programům poskytuje rozhraní pro dotazování. Stub resolver odesílá dotazy na DNS recursor nebo Forwarder, které pro něj získají odpověď.

1.5 DNS záznamy

Pro jednotlivé domény je v DNS zaznamenána pestrá směs různorodých údajů, které se nazývají zdrojové záznamy (resource records, RR) a jsou specifikovány v RFC 1035. [45]. V následující tabulce jsou uvedeny položky, které obsahuje jeden záznam.

NAME	Jméno, ke kterému se záznam vztahuje.
TYPE	Typ informace nesené v tomto záznamu.
CLASS	Rodina protokolů pro které je záznam určen.
TTL	Doba, po kterou je možné mít záznam uložený v cache paměti.
RDLENGTH	Délka dat položky RDATA v oktetech.
RDATA	Datový obsah záznamu.

Tabulka 1.1: Položky zdrojového záznamu (RR, Resource Record)

Interpretace a struktura pole RDATA je závislá na hodnotě pole TYPE. Pro jedno jméno zpravidla existuje několik záznamů, přičemž může dojít ke kombinaci různých typů, ale i opakovaných výskytů stejného typu. Ty nejzákladnější typy záznamů jsou popsány v následující tabulce. [3]

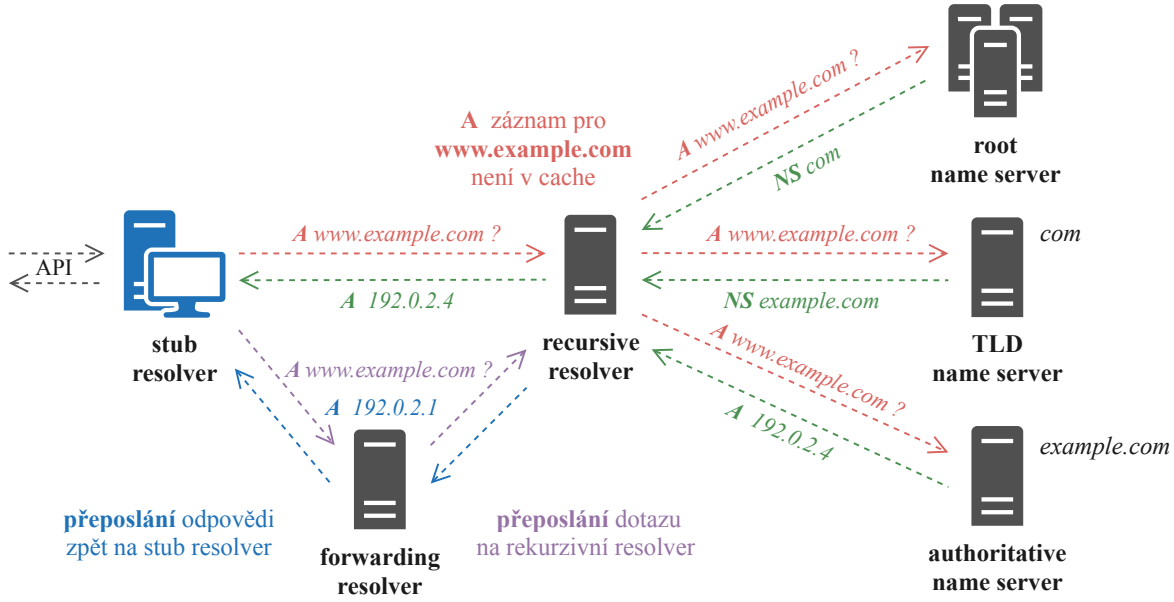
A	*IPv4 Address*, IPv4 adresa
AAAA	*IPv6 Address*, IPv6 adresa
NS	*NameServer*, Autoritativní name server pro danou doménu.
CNAME	*CanonicalName*, Mapuje doménu nebo subdoménu na jinou doménu.
MX	*Mail eXchanger*, Mail server pro danou doménu.
PTR	*PoinTeR*, Doménové jméno pro reverse-lookup.
SOA	*Start Of Authority*, Základní informace o doméně.
TXT	*Text*, Administrátorovy poznámky.

Tabulka 1.2: Základní typy DNS záznamů

Soubory, ve kterých jsou zdrojové záznamy dané zóny uloženy se nazývají zónové soubory (zone files). Tyto soubory jsou v textové podobě a mimo zdrojové záznamy mohou obsahovat komentáře nebo direktivy používané k ovládání zpracování zónového souboru. [3]

1.6 DNS dotaz

DNS protokol je postaven na principu klient-server. V jednoduchých případech komunikace zahrnuje výměnu pouhých dvou paketů, kdy klient pošle dotaz a server na něj odpoví. Jako základní transportní protokol je používán UDP. TCP se používá v případech, kdy je objem přenášených dat větší. Pro UDP i TCP se na serverové straně standardně používá port 53. [41]



Obrázek 1.2: Hierarchie DNS serverů a princip DNS dotazu

Vysvětlení na příkladu

1. Uživatel zadal do svého webového prohlížeče `www.example.com` a prohlížeč se pomocí *API* zeptá na tuto adresu stub resolveru přímo v operačním systému.

2. Stub resolver se obrátí na rekurzivní nebo forwardující resolver s dotazem na IP adresu pro `www.example.com`.

3. Pokud dorazí dotaz k forwardujícímu resolveru, je přeposlán na rekurzivní resolver.

4. Rekurzivní resolver nemá požadovanou informaci v cache paměti, pokud by jí měl, hned by mohl odpovědět. Resolver tedy informaci nezná, zná však adresy všech kořenových jmenných serverů a jednoho z nich se zeptá.

5. Kořenový server odpověď také nezná, ale ví, že existuje TLD doména `com` a zná IP adresy pro její autoritativní jmenné servery, které pošle zpět rekurzivnímu resolveru.

6. Rekurzivní resolver obdrží adresy jmenných serverů ke `com` doméně a pošle na jeden z nich dotaz na IP adresu `www.example.com`.

7. Oslovený server opět záznam nezná, ale zná adresy jmenných serverů pro `example.com` a ty rekurzivnímu resolveru poskytne.

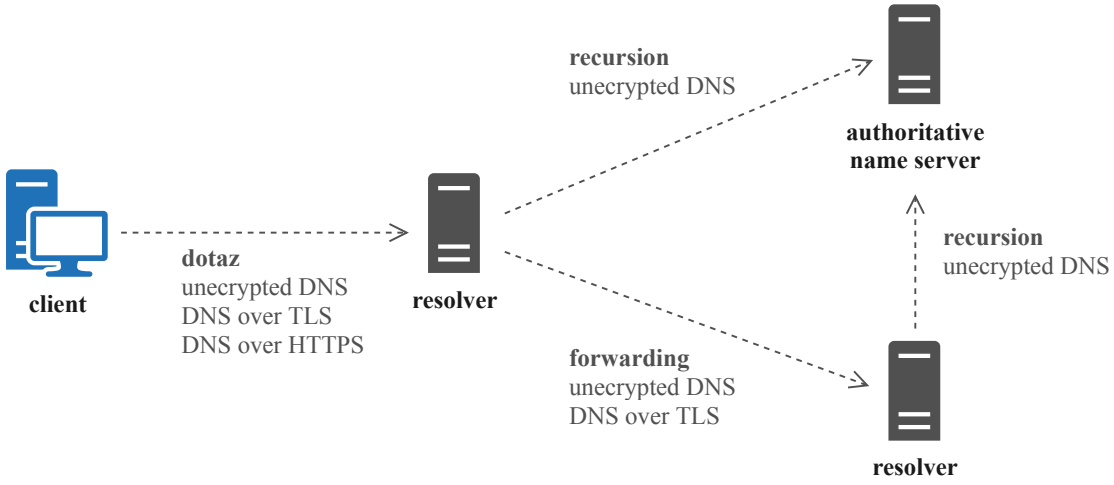
8. Resolver opět jednu z adres jmenných serverů pro doménu `example.com` vybere a pošle dotaz na IP adresu `www.example.com`

9. Toto jméno se již nachází v doméně `example.com` a tak jmenný server odpoví IP adresou pro `www.example.com`.

10. Rekurzivní resolver obdrží výslednou odpověď a přepošle ji tazateli (stub/forwarding resolver), ten zase svému tazateli, dokud se odpověď nedostane až k místu původu dotazu.

1.7 Zabezpečení komunikace

V základu je komunikace mezi DNS servery a klienty bez jakéhokoliv zabezpečení. Každý může vidět obsah posílaných dotazů a odpovědí, případně měnit jejich obsah. Přes DNS resolvery, provozované například poskytovateli internetového připojení (ISP), protéká obrovské množství dotazů a tak není na jeho výstupu možné jednoznačně určit odesílatele dotazu. Jednoduše řečeno samotná architektura DNS poskytuje svým klientům základní úroveň ochrany soukromí. [21, 35]



Obrázek 1.3: Zabezpečení komunikace

Na obrázku 1.3 je možné vidět, kde se mezi klientem a servery používá zabezpečení komunikace. K zabezpečení komunikace se standardně používá protokol DNS over TLS a nebo relativně nový protokol DNS over HTTPS.

1.7.1 DNS over TLS

DNS over TLS (DoT) je bezpečnostním protokolem pro šifrování DNS dotazů a odpovědí pomocí protokolu TLS (Transport Layer Security). Na rozdíl od základního DNS probíhá komunikace na portu 853. DoT je standardizováno v RFC 7858 [29].

1.7.2 DNS over HTTPS

DNS over HTTPS (DoH) je navrhovaným standardem, který byl publikován v říjnu roku 2018 jako RFC 8484 [28]. Tento standard definuje protokol pro odesílání DNS dotazů a přijímání DNS odpovědí přes HTTPS protokol, tedy na portu 443. Za DoH klienta je nejčastěji označován webový prohlížeč.

V současné době má DoH i celou řadu kritiků, kteří tvrdí, že DoH poskytuje falešný pocit bezpečí, protože šifruje pouze informace, které mohou být stále získány prostřednictvím nešifrovaných částí HTTPS požadavků, jako jsou IP adresy a označení serveru. Implementace DoH v prohlížečích se v současné době spoléhají na implementace DoH DNS třetích stran (Google, Mozilla), což je přímo v rozporu s decentralizovanou povahou DNS a může to mít negativní dopad na soukromí uživatelů. [30] DoH může také bránit analýze a sledování provozu DNS pro účely kybernetické bezpečnosti, kdy například červ *Godlua* použil na DDoS (Distributed Denial of service) útok DoH k maskování připojení ke svému řídícímu serveru. [17]

1.8 DNSSEC

Když například DNS resolver obdrží odpověď na dotaz A záznamu webové stránky `www.example.com`, může pouze doufat, že data jsou správná. Obdržená IP adresa může být podvržená, přičemž uživatel v prohlížeči nezaregistruje žádnou změnu. V případě, že `www.example.com` bude například stránka internetového obchodu, může při platbě útočník získat údaje kreditní karty a i tak se pro uživatele vše může jevit jako oficiální stránka obchodu. Domain Name System Security Extensions (DNSSEC) je sada specifikací definující proces, kterým může správně nakonfigurovaný DNS resolver ověřit původ, pravost a integritu dat odpovědi z podepsané zóny. DNSSEC je definován ve specifikacích RFC 4033, 4034 a 4035 [4, 6, 5] a dále je rozšířen o RFC 4470, 4509, 5011 a 5155 [57, 25, 50, 36]. DNSSEC také zavádí několik nových DNS záznamů, RRSIG, DNSKEY, DS a NSEC3. [3]

Díky DNSSEC může DNS klient pomocí elektronického podpisu ověřit původ, platnost a integritu dat. Držitel domény, která podporuje DNSSEC, si vygeneruje dvojici privátního a veřejného klíče. Svým privátním klíčem podepíše technické údaje, které o své doméně do DNS vkládá. Pomocí veřejného klíče je pak možné ověřit pravost tohoto podpisu. Aby byl veřejný klíč opravdu veřejný a dostupný všem, je publikován držitelem domény u jeho nadřazené domény. Vytváří se tak řetězec důvěry (Chain of Trust), který zajišťuje důvěryhodnost údajů. [3]

RRSET Základním krokem DNSSEC je seskupení jednotlivých záznamů RR (Resource Record) do balíčků zvaných RRSET. Tento balíček sdružuje všechny záznamy stejného typu a označení do jednoho, který je tak možné celý elektronicky podepsat. Díky tomu není potřeba podepisovat jednotlivé záznamy samostatně. Tím se snižuje provoz potřebný k ověření důvěryhodnosti záznamů, jelikož potřeba ověření záznamů stejného typu vzniká pouze jednou. [3, 42]

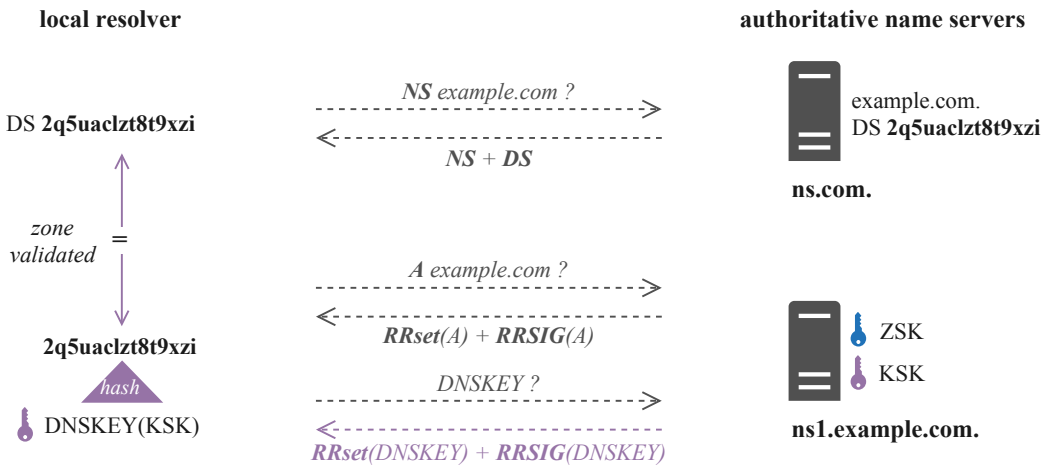
Zone Signing Key (ZSK) Jakmile jsou RRSETy vytvořeny, autoritativní jmenný server této zóny každý RRset podepíše pomocí *ZSK (Zone Signing Key)*. Jedná se o asymetrický způsob kryptografie, kdy jsou RRSETy včetně ZSK podepsány pomocí privátní části ZSK. Výsledný podpis je pak uložen jako samostatný záznam s názvem RRSIG (Resource Record Signature). K ověřování podpisu je používána veřejná (public) část ZSK, která je zpřístupněna jako další samostatný záznam DNSKEY. Pokud se klient zeptá například na A záznam pro `example.com`, dostane od autoritativního serveru odpověď v podobě A záznamu a RRSIG záznamu. Zeptá znovu na DNSKEY záznam pro `example.com` a poté může ověřit pravost záznamů a důvěryhodnost zdroje. [3, 42]

Key Signing Key (KSK) U ZSK vzniká problém pokud se útočníkovi podaří mimo klasického záznamu podvrhnout i odpověď s veřejným klíčem DNSKEY. Proto je potřeba nejprve ověřit pravost veřejného klíče ZSK. K tomuto účelu slouží další dvojice klíčů KSK (Key Signing Key). Ověřování ZSK pomocí KSK probíhá obdobně jako u ověřování pravosti RRsetů. KSK podepisuje veřejnou část ZSK (DNSKEY) a tím vznikne další RRSIG záznam. Veřejná část KSK je pak uložena obdobně do DNSKEY záznamu, která společně s DNSKEY pro ZSK tvoří RRset záznamů typu DNSKEY. [3, 42]

Delegation Signer (DS) Díky ZSK a KSK je možné věřit pravosti záznamů, nyní je ale potřeba ověřit pravost KSK. K tomu slouží záznam DS (Delegation Signer), který je uložen u nadřazené zóny, čímž vzniká jakýsi řetězec důvěry (Chain of Trust). Kdykoli je nastavena podřízená zóna, je její nadřazené zóně předána hash kopie veřejného klíče KSK této zóny publikována jako DS záznam. Pokaždé když se klient zeptá na podřízenou zónu dostane tento záznam. To navíc klientovi dá informaci zda tato podřízená zóna podporuje DNSSEC. Jediné co musí klient udělat je vytvořit hash KSK podřízené zóny a porovnat ji s DS záznamem a tak ověřit důvěryhodnost zóny. [3, 42]

1.8.1 Postup validace

Zde je popsán zjednodušený postup validace záznamu pomocí DNSSEC pouze pro získání představy k čemu který záznam slouží. Na schématu 1.4 je znázorněna komunikace mezi servery pro získání potřebných dat k provedení validace pomocí DNSSEC a na schématu 1.5 je znázorněn proces ověření.



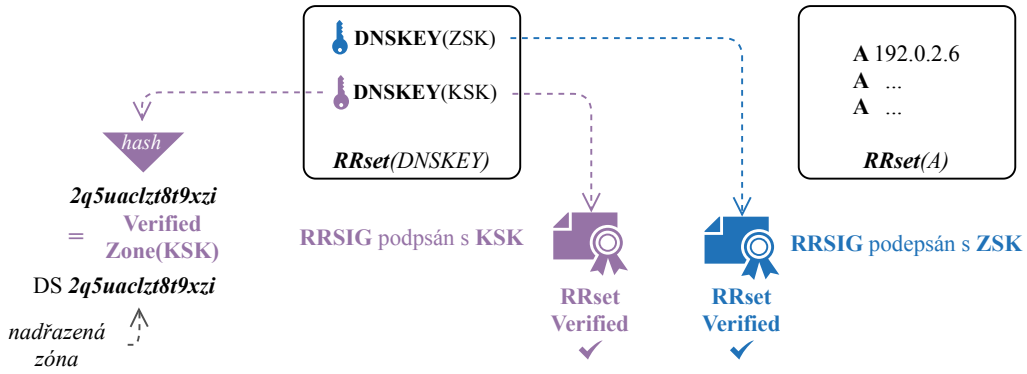
Obrázek 1.4: DNSSEC komunikace

1. Resolver se zeptá jmenného serveru na A záznam `example.com`.
2. Jmenný server vrátí RRset(A) obsahující RRSIG(A) podpis RRsetu pomocí ZSK.
3. Resolver se poté dotáže na DNSKEY záznam, tedy veřejný klíč k ZSK.
4. DNS server vrátí RRset(DNSKEY) obsahující ZSK i KSK a také RRSIG(DNSKEY) podpis RRsetu pomocí KSK.
5. Resolver má nyní vše co potřebuje k ověření, že data byla podepsána správným klíčem. Nemůže ovšem ověřit zda se jedná o důvěryhodný klíč.
6. Požádá proto nadřazený jmenný server o NS záznam pro požadované jméno `example.com`.
7. Nadřazený jmenný server společně s odpovědí NS pošle i DS záznam pro tuto doménu.

8. Resolver vypočte hash z DNSKEY(KSK) záznamu obdrženého v RRset(DNSKEY) a porovná jej s DS záznamem obdženým od nadřazené autority. Pokud se shodují je možné name serveru věřit.

9. Resolver tedy má ověřený KSK, kterým může ověřit podpis RRsetu(DNSKEY).

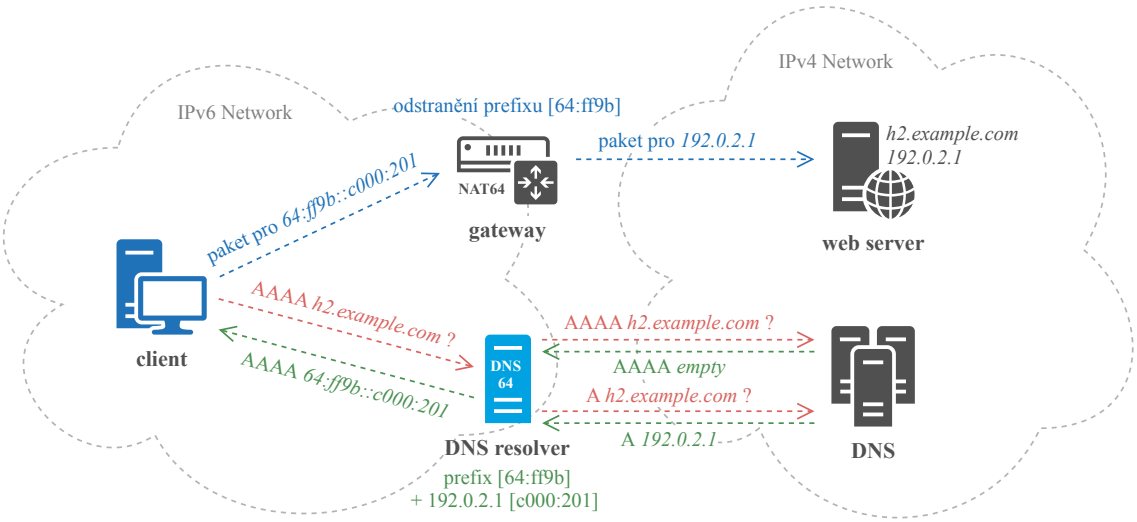
10. Díky ověření podpisu RRsetu(DNSKEY) je ověřený i ZSK, kterým je podepsán RRset(A) a tím validovat důvěryhodnost A záznam pro `example.com`.



Obrázek 1.5: DNSSEC detailní validace

1.9 DNS64

DNS64 je rozšíření pro rekurzivní DNS servery popsané v *RFC 6147* [7]. Společně s *NAT64* se nejčastěji používá s cílem umožnit zařízením v koncové IPv6 síti přístup ke službám poskytovaným pouze protokolem IPv4. NAT64 je rozšířením NAT (Network Address Translation) pro IPv6 adresy a umožňuje mapování mezi IPv4 a IPv6 adresami. Základem je použití jednotného prefixu pro mapování IPv4 adres na IPv6 v NAT64 a DNS64. Nejčastěji jde o dobře známý `64:ff9b::/96` prefix popsaný v *RFC 6052*, nebo o prefix délky maximálně 96b definovaného správcem sítě. Například u délky prefixu 96b je IPv4 adresa připojena přímo za prefix, podobu mapování udává tabulka v RFC 6052 [8].



Obrázek 1.6: Princip DNS64 společně s NAT64

DNS64 se používá při dotazování na záznamy typu AAAA, čili na IPv6 adresy pro zadaná jména. Princip je vizualizován na obrázku 1.6 a popsán následovně:

1. Klient v IPv6 síti se dotáže DNS resolveru s podporou DNS64 na AAAA záznam požadovaného serveru.
2. DNS resolver hledá informaci buď ve vyrovnávací paměti nebo provede rekurzivní dotazování na jmenné servery (kapitola 1.6).
3. Pokud ani poté nedostane AAAA záznam, zeptá se na A záznam.
4. Po obdržení A záznamu je AAAA záznam vytvořen spojením obdrženého A záznamu s IPv6 prefixem podle RFC 6052 [8] a výsledek je odeslán klientovi.
5. Klient získá odpověď od rekurzivního DNS v podobě IPv6 adresy a pošle potřebný paket na tuto adresu.
6. Žádost dorazí k bráně (gateway), která podporuje NAT64. Zde je zpětně z IPv6 adresy odebrán prefix a paket je přeposlána na výslednou IPv4 adresu.
7. Server v IPv4 síti obdrží žádost.

1.10 RPZ

Response policy zone (RPZ) je metoda, která umožňuje rekurzivnímu DNS resolveru vrátit DNS klientovi upravenou odpověď na jeho dotazy. K vyjádření úprav DNS odpovědí slouží speciálně konstruované zóny. Takto upravené odpovědi mohou například bránit k přístupu na vybrané domény, přesměrovat uživatele nebo blokovat nežádoucí e-mail. Proto se také tyto "DNS firewally" hojně využívají k boji proti internetové kriminalitě. [56]

Uživatel používá rekurzivní DNS resolver, který je nakonfigurovaný pro použití RPZ a v některém z jeho souborů má web `bad.example.com` označený jako nebezpečný. Uživatel obdrží email ve kterém je odkaz na již zmíněnou webovou adresu `bad.example.com`, která se tváří důvěryhodně, klikne na odkaz a pošle dotaz na rekurzivní resolver. Ten ale již ví, že tato stránka není důvěryhodná a tak na místo zaslání adresy nebezpečného webu přesměruje uživatele na informativní stránku o tom co se stalo. [56]

```
$ORIGIN rpz.example.net.
$TTL 1H
@                   SOA LOCALHOST. named-mgr.example.net. (
                                        1 1h 15m 30d 2h)
                    NS LOCALHOST.

; QNAME policy records.
; There are no periods (.) after the relative owner names.
nxdomain.example.com        CNAME   .       ; NXDOMAIN policy
nodata.example.com          CNAME   *.      ; NODATA policy

; Redirect to walled garden
bad.example.com             A       10.0.0.1
                           AAAA    2001:db8::1
```

Zdrojový kód 1.1: Ukázka RPZ zónového souboru souboru [56]

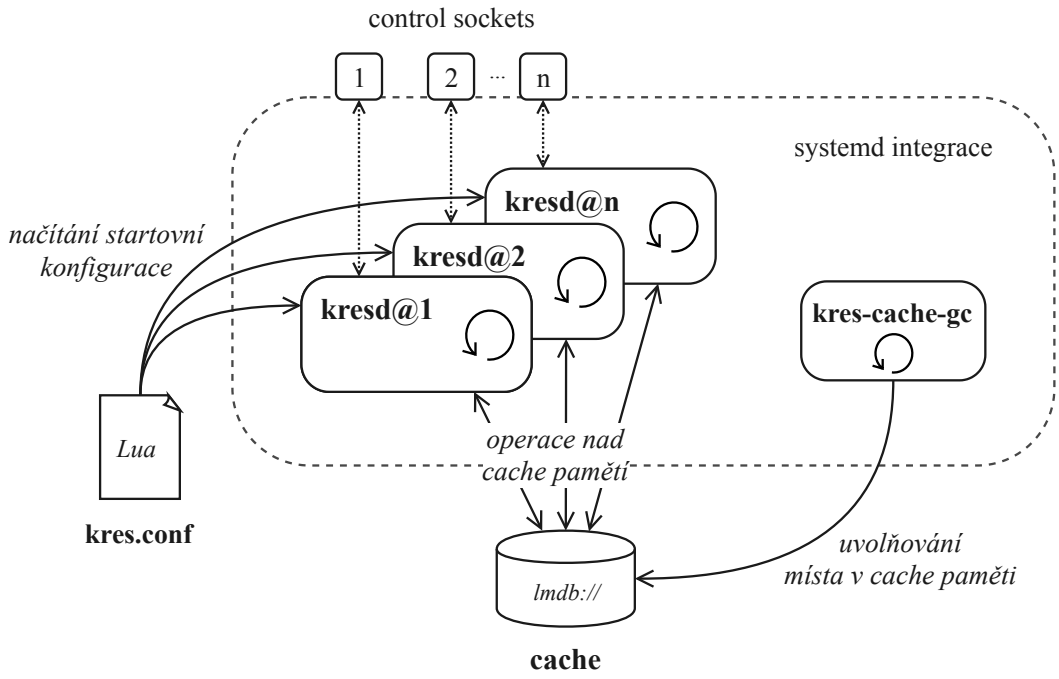
2 Knot Resolver

Knot Resolver je open-source implementací takzvaného caching full DNS resolveru. Knot Resolver patří do skupiny projektů Knot, kterou v roce 2010 započal projekt Knot DNS jakožto open-source implementace autoritativního DNS serveru. Účelem tohoto projektu je poskytnout alternativní open-source řešení autoritativního DNS serveru vhodného pro TLD s cílem zvýšení celkové bezpečnosti, stability a odolnosti DNS. V roce 2015 následovalo představení projektu Knot Resolver, který je spolu s ostatními projekty rodiny Knot aktivně vyvíjen v laboratořích sdružení *CZ.NIC, z.s.p.o.* [18], které je správcem české domény `.cz`. [35, 39, 20]

2.1 Architektura

Knot Resolver se vyznačuje modulární architekturou, která udržuje jádro resolveru napsané v jazyce C malé a efektivní. Klíčovou vlastností oproti ostatním open-source implementacím je state-machine API pro tvorbu rozšíření a modulů, na jejichž tvorbu je možné použít jazyk C nebo Lua. Díky této architektuře Knot Resolver umožňuje pokrýt nepřebernou škálu použití, od malý kanceláří až po velké poskytovatele internetu ISP (Internet Service Provider). [35]

Knot Resolver se také skládá z několika samostatných jedno-vláknových procesů, samotného resolvujícího daemona `kresd` a údržbáře cache paměti `kres-cache-gc`. Veškeré samostatné procesy Knot Resolveru využívají integraci se systemd, která poskytuje jednoduchou správu nad procesy. Následující schéma (obrázek 2.1) představuje jednotlivé části a procesy, které jsou detailněji popsány v následujících kapitolách.



Obrázek 2.1: Architektura Knot Resolveru

2.2 Daemon (kresd)

Jde o jedno-vláknový proces schopný běžet pouze na jednom procesoru/jádře. Aby bylo možné vytížit více procesorů/jader je `kresd` proces možné spustit jako několik instancí zároveň sdílejících společnou cache paměť. Všechny instance mohou poslouchat na stejném portu díky možnosti `SO_REUSEPORT` u síťových soketů [34], kdy jádro operačního systému (kernel) rozděluje příchozí dotazy mezi jednotlivé procesy automaticky. Velkou výhodou této architektury je, že pokud dojde k chybě v jednom z těchto procesů, tato chyba nijak neovlivní ostatní procesy. Nedoje tak k chybě u ostatních procesů a k úplné nedostupnosti DNS služby. Chybový proces poté stačí restartovat a všechny ostatní procesy fungují jak mají beze změny. Na to navazuje další výhoda, kterou je možnost změny v konfiguraci aplikovat na jednotlivé instance postupně a tím v průběhu změny nemít nikdy plně vypnutý resolver nebo dokonce používat více instancí s jinou konfigurací. [35, 39]

Moduly Mimo samotné resolvující jádro závisí funkcionalita Knot Resolveru na oddělených modulech. Ve skutečnosti je i samotné jádro Knot Resolveru tvořeno pomocí základních modulů (iterator, validator, cache), které jsou ve výchozím nastavení načítány automaticky a samotný daemon bez modulů řeší pouze komunikaci po síti a správu konfigurace. To umožňuje kombinací modulů dosáhnout požadované funkcionality bez zpomalení výkonu nepotřebnými funkcemi. Modul je před použitím a konfigurací je potřeba každý jednotlivý modul nejdříve načíst. Příkladem může být načtení a nastavení modulu pro použití DNS64 (kapitola 1.9) na příkladu konfigurace (zdrojový kód 2.1).

2.3 Cache paměť

Cache paměť Knot Resolveru je realizována pomocí LMDB (Lightning Memory-Mapped Database), databáze mapovaná do pamět uložena na disku. Díky její persistenci jednotlivé instance `kresd` sdílející cache paměť neztrácí data při restartu nebo selhání. Knot Resolver dále implementuje tzv. agresivní použití cache paměti pro data ověřená pomocí DNSSEC (RFC 8198 [24]), to ve výsledku zvyšuje výkon a také chrání před některými typy útoků pomocí náhodných subdomén. [35]

2.3.1 Cache Garbage Collector (kres-cache-gc)

Knot Resolver, konkrétně instance jeho `kresd` procesu, používá dostupnou cache paměť dokud není plná, pokud se tak stane a je vyžadováno další místo je celá cache vyprázdněna. Jelikož má cache silný vliv na celkový výkon DNS resolveru nejedná se o optimální řešení. Aby nedocházelo k opětovnému startu resolveru s prázdnou cache pamětí, je k dispozici samostatný nezávislý proces `kres-cache-gc` nazývaný jako *cache garbage collector*, který se o uvolňování místa v cache stará průběžně

Namísto naprostého vyprázdnění cache paměti při jejím zaplnění, garbage collector uvolňuje místo postupně periodicky po určitém intervalu. V časovém intervalu zkontroluje využití cache paměti a pokud je dosažen určitý procentuální práh (threshold) zaplnění, pokusí se na základě heuristického ohodnocení jednotlivých záznamů uvolnit požadované místo.

2.3.2 Konfigurace

Konfigurace `kresd` daemona, je ve skutečnosti skript se syntaxí programovacího jazyka Lua, který obsahuje funkce a proměnné měnící parametry uvnitř procesu. Díky tomu je možné při psaní konfigurace využít plný potenciál tohoto programovacího jazyka a psát značně komplexní pravidla pro konfiguraci. Naopak konfigurace cache garbage collectoru je možná pouze pomocí parametrů při jeho spuštění z příkazové řádky. Pro změnu nastavení je tedy potřeba proces zastavit a spustit znovu s novými parametry. [35]

```
-- this is a comment: listen for unencrypted queries
net.listen('192.0.2.1')

-- another comment: listen for queries encrypted using TLS on port 853
net.listen('192.0.2.1', 853, { kind = 'tls' })

-- 10 MB cache is suitable for a very small deployment
cache.size = 10 * MB

-- Load DNS64 module
modules.load('dns64')

-- Configure DNS64 with a NAT64 address
dns64.config('fe80::21b:aabb:0:0')
```

Zdrojový kód 2.1: Lua - příklad konfigurace Knot Resolveru

Konfigurační soubor Nejjednodušším způsobem, jak nakonfigurovat Knot Resolver je úprava konfiguračního souboru `/etc/knot-resolver/kresd.conf`. Při spuštění jedné nebo více `kresd` instancí pomocí systemd je tento konfigurační soubor automaticky načten a aplikován. Pokud není systemd k dispozici je možné při spuštění `kresd` z příkazové řádky přidat cestu ke konfiguračnímu souboru jako vstupní parametr. V případě, že je potřeba upravit nastavení již běžící instance `kresd`, musí se konfigurační soubor upravit a resolver restartovat. [35]

Run-time konfigurace Běžící instance `kresd` je možné konfigurovat za běhu pomocí Lua příkazů posílaných přes *Unix domain socket*, přičemž každá `kresd` instance má svůj vlastní řídící socket. V dalším případě je možné spustit `kresd` v interaktivním módu, kde lze LuaJit příkazy zadávat přímo. Při použití těchto dvou způsobů je možné i číst konfiguraci nebo statistiky jednotlivých instancí. [35].

3 NETCONF

Network Configuration Protocol známý spíše pod zkratkou NETCONF je protokol navržený organizací IETF a určený pro konfiguraci síťových zařízení. Nejedná se pouze o nastavování různých parametrů daného zařízení, čtení konkrétních konfiguračních a stavových informací, ale i o provádění vzdálených operací nebo komplexních transformací v konfiguraci síťových zařízení. První specifikace byla publikována v roce 2006 jako RFC 4741 [22], poté následovalo několik rozšíření a následně byl v roce 2011 NETCONF protokol revidován v RFC 6241 [23]. Konfigurační data a zprávy NETCONF protokolu jsou kódovány v textové formě pomocí značkovacího jazyka XML (Extensible Markup Language).

3.1 Historie

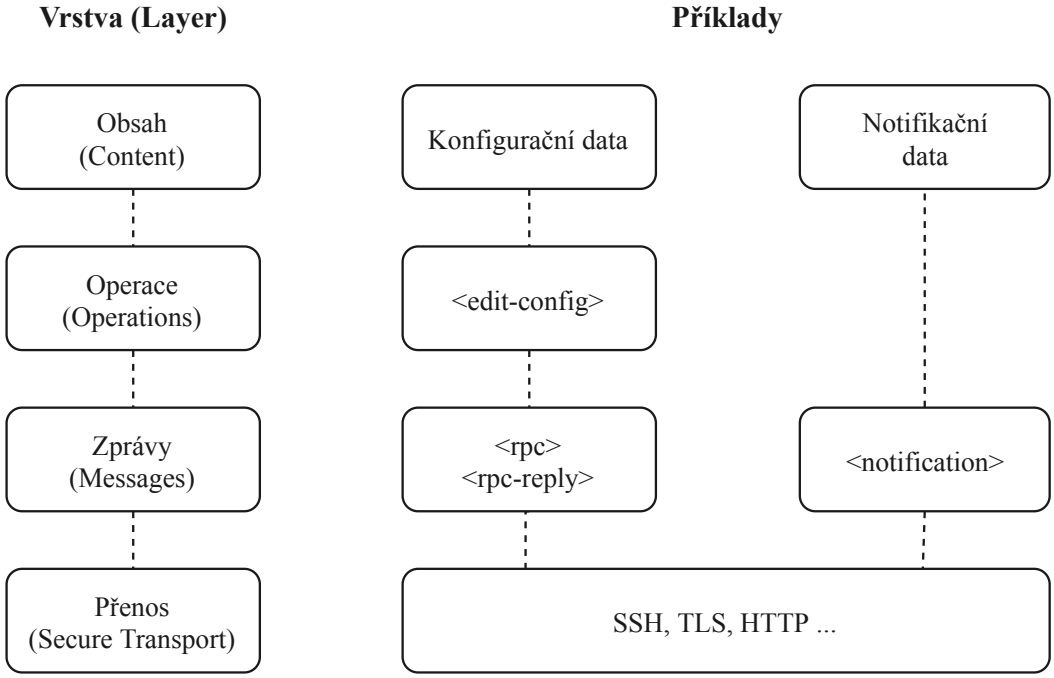
Koncem 80. let organizace IETF navrhla protokol SNMP (Simple Network Management Protocol), který se ukázal jako populární a začal se hojně používat. Na počátku 21. století se ale zjistilo, že SNMP již nebyl využíván ke konfiguraci síťových zařízení, ale jeho využití se přesunulo spíše k monitorování sítě.

V roce 2002 se rada *Internet Architecture Board* a klíčoví členové IETF komunity pro síťovou správu společně sešli s provozovateli sítí, aby projednali tehdejší situaci. Výsledkem této schůzky bylo RFC 3535 [48], ve které se ukázalo, že provozovatelé sítí primárně používali proprietární rozhraní přes příkazový řádek (CLI, Command Line Interface) namísto protokolu SNMP. Hlavním důvodem bylo, že výrobci zařízeních přidávali některé možnosti konfigurace pouze do jejich proprietárních CLI, tudíž řada z funkcionalit nebyla dostupná přes SNMP a tedy zařízení nebylo možné s jeho pomocí kompletně nakonfigurovat. Zároveň se také administrátorům v CLI osvědčilo používání příkazů jednoduché textové podobě oproti SNMP, které bylo založené na kódování BER a relativně komplikované. [48]

Přibližně ve stejnou dobu Juniper Networks používali k řízení své sítě přístup založený na jazyku XML, který byl prezentován širší komunitě a návrh budoucího protokolu je na něm do jisté míry založen. Tyto události vedly IETF v roce 2003 k vytvoření pracovní skupiny s názvem NETCONF. Skupina byla pověřena prací na novém protokolu, který v roce 2006 zveřejnila.

3.2 Architektura

Veškeré operace NETCONF protokolu, tedy i změna konfigurace, jsou prováděny pomocí *vzdáleného volání procedur* RPC (Remote Procedure Call). Architektura NETCONF protokolu je založena na čtyřech logických vrstvách, které jsou naznačeny na obrázku 3.1. [23]



Obrázek 3.1: Vrstvy NETCONF protokolu [23]

Přenos (Transport) Úkolem transportní vrstvy je zprostředkovat spojení mezi konfigurovaným zařízením (netconf-server) a administrátorem (netconf-client). Mezi nejvyužívanější protokoly této vrstvy patří SSH, ale zprostředkovatelem může být jakýkoli protokol splňující požadavky definované v NETCONF specifikaci.

Zprávy (Messages) Tato vrstva poskytuje jednoduchý mechanismus pro kódování RPC operací a notifikací, který je nezávislý na způsobu přenosu dat. NETCONF využívá jednoduchého principu RPC, kdy klient (netconf-client) vytvoří požadavek na provedení operace `<rpc>` a tento požadavek je odeslán transportní vrstvou serveru (netconf-server). Server se pokusí požadovanou operaci vykonat a následně odešle klientovi výsledek požadované operace `<rpc-reply>`. Notifikace `<notification>` jsou na rozdíl od RPC asynchronní a jsou zasílány pouze serverem směrem ke klientovi.

Operace (Operations) Vrstva operací definuje sadu operací volané jako RPC metody s parametry kódovanými v XML. Protokol poskytuje pouze základní sadu operací pro konfiguraci zařízení. Díky možnosti rozšíření mohou síťová zařízení poskytovat i další specifické operace, například podle typu zařízení.

Obsah (Content) Tato vrstva jako jediná není specifikována NETCONF protokolem. Obsahuje například konkrétní konfigurační data jejichž forma a význam je definována výrobcem zařízení nebo softwaru.

Modelovací jazyk YANG (kapitola 3.3) byl vytvořen pro specifikaci NETCONF datových modelů a operací, které pokrývají poslední dvě vrstvy (obsah, operace) architektury NETCONF protokolu. [23]

3.3 YANG

YANG (Yet Another Next Generation) je jazyk pro modelování dat navržený přímo pro definici dat posílaných pomocí protokolů síťové správy NETCONF a RESTCONF. Tento jazyk je udržovaný organizací IETF, konkrétně pracovní skupinou NETMOD [46], publikován byl v roce 2010 v RFC 6020 [11]. Nová verze YANG 1.1 vyšla v roce 2016 jako RFC 7950 [10]. YANG lze použít k modelování konfiguračních a stavových dat, umí definovat formát a parametry notifikací a RPC operací. Jazyk je nezávislý na protokolu a může být poté převeden do libovolného formátu, například XML nebo nově JSON (RFC 7951 [38]), které podporují protokoly NETCONF a RESTCONF. [10, 11]

YANG je modulární jazyk, datový model je definovaný pomocí zdrojových souborů neboli YANG modulů. Tyto moduly umožňují definovat hierarchické datové struktury, které je možné mezi sebou kombinovat a rozšiřovat. K určení konkrétních elementů v datovém stromě je používán dotazovací jazyk XPath. Jazyk YANG definuje několik základních datových typů (`typedef`), které je možné použít pro odvození a upřesnění dalších datových typů specifických pro vytvářený datový model. Existuje několik základních druhů uzlů (nodes) definujících výslednou datovou strukturu. Tyto uzly lze také spojovat do skupin (`grouping`), které je možné následně použít na více místech v datovém schématu. [10, 38]

- **leaf**

  Koncový datový uzel, který v datovém stromě existuje nejvýše jednou a drží hodnotu.

- **container**

  Vnitřní datový uzel, který v datovém stromě existuje nejvýše jednou. Nemá hodnotu, ale sadu podřízených uzlů.

- **list**

  Vnitřní datový uzel, který může v datovém stromě existovat ve více unikátních instancích. Nemá hodnotu, ale sadu podřízených uzlů definující strukturu instance.

- **leaf-list**

  Koncový datový uzel, který definuje set unikátně identifikovatelných uzlů. Každý tento uzel má hodnotu, ale nemá podřízené uzly.

- **rpc**

  Specifická RPC operace. Nemá hodnotu, ale sadu podřízených uzlů definující vstupní a výstupní parametry operace.

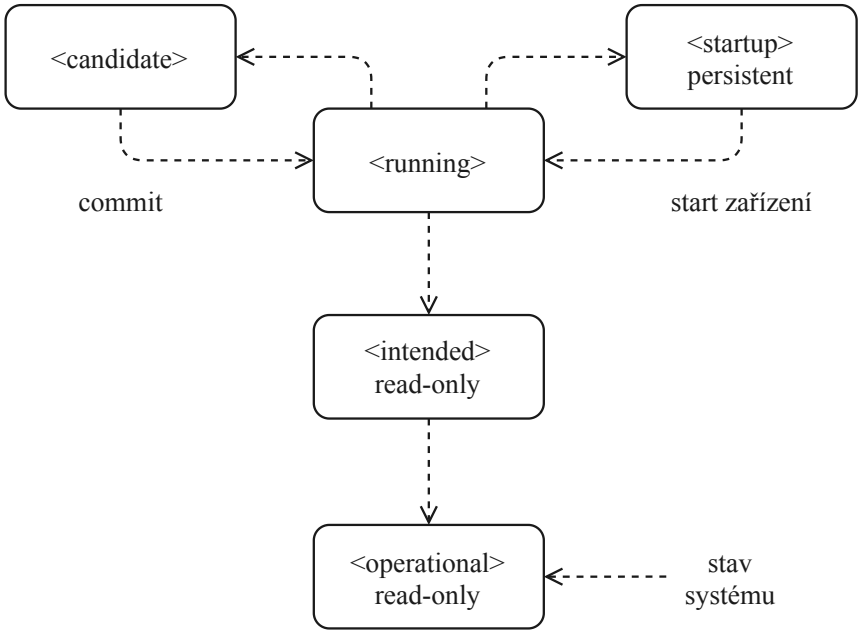
- **action**

  Operace definovaná pro konkrétní uzel v datovém stromě. Nemá hodnotu, ale sadu podřízených uzlů definující vstupní a výstupní parametry operace.

3.4 Datová úložiště (datastores)

Datová úložiště jsou základním konceptem aplikace datových modelů pro protokoly správy síťových zařízení. Samotný NETCONF protokol definuje pro data tři základní datová úložiště (datastores), `<running>, <startup> a <candidate>`. [23] Tato původní architektura úložišť byla později aktualizována novou architekturou, která má zkratku NMDA (Network Management Datastore Architecture) RFC 8342 [14].

NMDA (Network Management Datastore Architecture) Jedná se o koncepční rámec architektury datových úložišť, který je specifikován v RFC 8342 [14]. Je přímo určen pro protokoly správy síťových zařízení jako je RESTCONF a jazyk pro modelování dat YANG. V roce 2019 vyšla rozšiřující specifikace pro podporu této nové architektury v NETCONF protokolu RFC 8526 [13]. NMDA aktualizuje YANG specifikaci RFC 7950 [10] a to především ve vztahu k nové architektuře datových úložišť nahrazující původní architekturu datových úložišť. Původní tři datová úložiště jsou rozšířeny o dvě nová `<intended>` a `<operational>`. [14]



Obrázek 3.2: Architektura datových úložišť podle NMDA [14]

- **startup**

  Jedná se o konfigurační datové úložiště, ze kterého si zařízení načítá konfiguraci při startu do úložiště `<running>`. Slouží k perzistentnímu uchování konfigurace napříč restarty zařízení.

- **candidate**

  Jedná se o konfigurační datové úložiště, které zrcadlí konfigurační data uložená v datovém úložišti `<running>` a lze s nimi manipulovat aniž by byla ovlivněna konfigurace zařízení. Lze jej použít k přípravě konfigurace, která je následně přesunuta do `<running>` úložiště.

- **running**

  Jedná se o konfigurační datové úložiště, které drží aktuální konfiguraci zařízení. Toto datové úložiště přímo ovlivňuje konfiguraci zařízení.

- **intended**

  Jedná se o datové úložiště určené pouze pro čtení představujícíe konfiguraci po provedení veškerých transformací. Obsah úložiště je konfigurace, kterou se snaží zařízení použít. Je pevně propojeno s `<running>` a kdykoli jsou na `<running>` zapsána nová data musí se okamžitě aktualizovat a validovat. Pro některé implementace NMDA může být `<intended>` a `<running>` úložiště totožné.

- **operational**

  Stejně jako `<intended>` je i `<operational>` datové úložiště pouze pro čtení. Jako jediné obsahuje jak konfigurační tak stavová data. Obsahuje stav systému a veškerou konfiguraci (operační) právě používanou zařízením, to zahrnuje aplikovanou konfiguraci z `<intended>` úložiště, konfiguraci poskytnutou systémem a nebo výchozí hodnoty definované datovým modelem.

3.5 RESTCONF

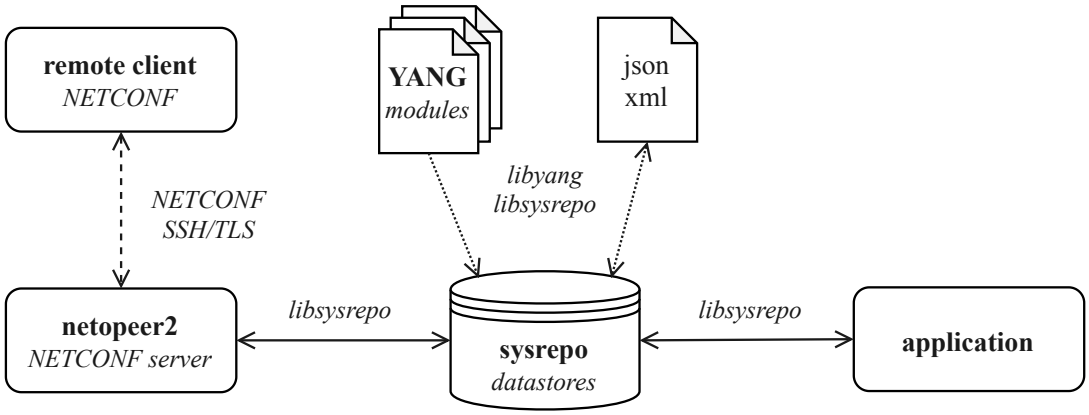
RESTCONF je protokol specifikovaný v RFC 8040 [9] umožňuje konfiguraci síťových zařízení prostřednictvím HTTP protokolu. Používá koncept datových úložišť z NETCONF protokolu a jazyk YANG pro definici datového modelu.

4 Použité technologie

V této kapitole jsou velmi stručně představeny použité technologie a nástroje. Související detaily jsou vysvětleny v kontextu návrhu a realizace v jednotlivých kapitolách práce.

4.1 sysrepo

Jedná se o konfigurační a provozní datové úložiště pro Unixové/Linuxové aplikace vyvíjené spolu s nástroji `netopeer2` a `libyang` ve sdružení *CESNET, z.s.p.o.*. `sysrepo` implementuje datová úložiště NETCONF protokolu (kapitola 3), která mohou aplikace využít pomocí knihovny `sysrepo` ke správě své konfigurace. Konfigurace je definována pomocí modelovacího jazyka YANG. `sysrepo` zajišťuje konzistenci dat v datovém úložišti a vynucuje pravidla definované pomocí YANG datového modelu. S pomocí `netopeer2` serveru se datové úložiště a tím pádem i aplikace stávají vzdáleně konfigurovatelné. [16]



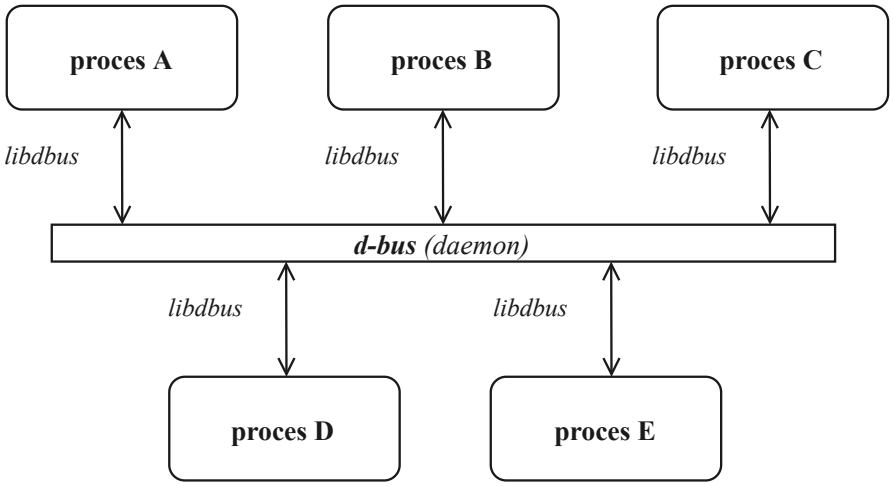
Obrázek 4.1: sysrepo

Netopeer2 Jedná se o implementaci NETCONF serveru a pomocné sady nástrojů implementující síťové konfigurační nástroje založené na NETCONF protokolu (kapitola 3). Netopeer2 je založen na nové generaci NETCONF a YANG knihoven `libyang` a `libnetconf2`. `sysrepo` je použito jako implementace NETCONF datového úložiště (3.4). Společně tvoří kompletní implementaci NETCONF frameworku pro Linuxové distribuce. [16]

libyang Knihovna `libyang` je sada nástrojů pro modelování dat a práci s jazykem YANG (3.3). `sysrepo` používá libyang pro zpracování YANG schémat a dat. Knihovna je implementována v jazyce C pro GNU/Linux a poskytuje C API. [15]

4.2 D-Bus

D-Bus je softwarová sběrnice a mechanismus, který umožňuje komunikaci mezi více procesy (InterProcess Communication) běžící ve stejné instanci operačního systému. D-Bus byl vyvinut jako součást projektu `freedesktop.org`. Má dvě hlavní části, komunikační knihovnu `libdbus`, kterou mohou využít procesy pro komunikaci mezi sebou a `dbus` démona, který provozuje samotnou sběrnici přes kterou mohou procesy komunikovat právě s pomocí již zmíněné knihovny. [33]



Obrázek 4.2: Princip D-Bus

V jednom systému může být současně více aktivních instancí D-Bus sběrnice. Linuxové systémy běžně provozují dva druhy sběrnic. Jednu *systémovou sběrnici (system bus)* pro komunikaci v celém systému a *relační sběrnici (session bus)* uživatele, která přenáší provoz stejně jako sběrnice systémová, ale D-Bus si je vědom totožnosti uživatele a podporuje flexibilní mechanismy autentizace a řízení přístupu. [33]

4.3 systemd

Jde o sadu základních stavebních bloků pro operační systém Linux. Jeho hlavním cílem je sjednotit konfiguraci služeb a chování napříč linuxovými distribucemi. Primární složkou `systemd` je správce systému a služeb (services), který běží jako `PID 1` a spravuje ostatní služby systému. Pro spouštění těchto služeb používá například aktivaci sokety nebo D-Bus. Od roku 2015 jej přijala většina distribucí systému Linux. [52]

Jednou z důležitých komponent `systemd` je `watchdog`. Jak název napovídá jedná se o nástroj, který slouží k monitorování stavu běžící služby podporující tuto funkci. Musí být implementovaný přímo v kódu programu, kdy program sám `systemd` hlásí, že je vše v pořádku. Pokud se program určitou definovanou dobu neozve nebo sám ohlásí, že nastala chyba, `systemd` provede předem přijatá opatření. Typicky se jedná o restart služby nebo procesu. [53]

5 Problematika současného stavu

Tato část práce se zaměřuje na problematiku administrace Knot Resolveru, ale také na problematiku administrace různých implementací DNS resolverů obecně. Návrh nového systému je primárně zaměřen na administraci Knot Resolveru, nelze ovšem ignorovat obecné problémy při administraci DNS resolverů a proto se i pro některé z nich snaží tato práce navrhnout řešení. Toto je také důvodem proč je jako základ pro návrh systému využit standardizovaný protokol NETCONF (kapitola 3). Informace obsažené v této kapitole jsou výsledkem konzultací s vývojovým týmem Knot Resolveru, konzultantem práce a školitelem práce.

5.1 Administrace Knot Resolveru

Problematiku administrace Knot Resolveru řešenou v rámci práce je možné rozdělit do dvou hlavních částí. První se zaměřuje na okolnosti problematické administrace způsobené modulární architekturou Knot Resolveru a druhá část se soustředí přímo na formu konfigurace a konfiguračního souboru, kterou je jazyk Lua. V těchto okruzích jsou vysvětleny jednotlivé problémy a následně je navrženo teoretické řešení.

5.1.1 Modularita

Modulární architektura (kapitola 2.1) má své důvody, výhody a je svým způsobem mezi open-source implementacemi DNS resolveru unikátní, projevuje se ale negativně hned v několika oblastech administrace Knot Resolveru. Je náročnější řízení jednotlivých procesů a konfigurace jednotlivých procesů, kde je navíc možné určovat funkcionality na základě samostatných `kresd` modulů.

Řízení procesů Knot Resolver je realizován několika samostatnými procesy, které nejsou kromě `systemd` integrace nijak centrálně řízeny, je tedy nutné jednotlivé procesy spouštět manuálně podle potřeby a to i jednotlivé instance `kresd` procesu. Knot Resolver tak neposkytuje žádnou jinou možnost jak centrálně řídit všechny procesy nebo si nějakým způsobem přednastavit procesy a počet instancí ke spuštění.

Konfigurace procesů Podobně jako je potřeba řídit jednotlivé procesy Knot Resolveru samostatně, musejí být také samostatně konfigurovány. Jednotlivé instance `kresd` procesu je možné konfigurovat pomocí konfiguračního souboru společného pro všechny instance (kapitola 2.3.2), ale pouze při jejich startu. Pokud je potřeba upravit konfiguraci již běžících procesů, musí být konfigurován každý proces samostatně, nebo je upraven konfigurační soubor a jednotlivé procesy jsou restartovány. Proces `kres-cache-gc` je konfigurovatelný pouze při startu (kapitola 2.3.1). Jednáse o nepohodlný způsob konfigurace, ve kterém musí administrátor pro úpravu konfigurace udělat několik věcí zbytečně navíc a opakovaně.

Moduly kresd procesu Funkcionalita Knot Resolveru, přesněji řečeno procesu kresd (kapitola 2.2), je tvořena samostatnými moduly, které musejí být před použitím nebo úpravou jejich konfigurace nejdříve načteny, viz. zdrojový kód 2.1. Administrátor nejprve musí jednotlivé moduly načíst manuálně a až poté je může nakonfigurovat. V kombinaci se samostatnými instancemi procesů to způsobuje, že pokud chce administrátor rozšířit funkcionalitu pro všechny instance, musí načíst modul, provést konfiguraci a nakonec to samé udělat i u ostatních procesů.

Vyplývající požadavky:

- Sdílené úložiště pro konfiguraci Knot Resolveru a všech jeho procesů.
- Řídit, konfigurovat a monitorovat jednotlivé procesy by mělo být možné z jediného konfiguračního rozhraní CLI (Command Line Interface).
- Umožnit konfigurovat jednotlivé procesy i dynamicky za běhu a hromadně.
- Administrátor by se neměl starat o načítání jednotlivých modulů k procesu `kresd`, ale načítání modulů by mělo být automatické na základě konfigurace.

Návrh řešení Konfigurace Knot Resolveru bude spravována v sysrepo implementaci datového úložiště NETCONF protokolu a bude definována pomocí jazyka YANG. Z tohoto úložiště budou jednotlivé procesy odebírat svou konfiguraci pomocí knihovny sysrepo při startu a za běhu. Jednotlivé moduly pro `kresd` proces budou načítány automaticky na základě dostupné konfigurace nebo její změny. Řídící a jiné operace procesů budou definovány jako RPC operace nebo akce (YANG actions) taktéž v datovém modelu a bude je možné volat ze společného konfiguračního rozhraní. Konfigurační rozhraní bude používat také knihovnu sysrepo, ale k úpravě konfigurace v datovém úložišti, kterou procesy odebírají. Na řídící operace definované v datovém modelu bude reagovat nově navržený proces, který se bude starat o centrální řízení všech ostatních procesů realizujících Knot Resolver.

5.1.2 Lua konfigurace

Jednotlivé instance `kresd` procesu používají ke konfiguraci programovací jazyk Lua (kapitola 2.3.2). Na první pohled, především z pohledu komplexnosti, se může zdát použití plnohodnotného programovacího jazyka jako výhodné, pokud je ale na problematiku nahlíženo z pohledu validace konfigurace a použitelnosti pro uživatele a administrátory, je použití nevýhodné a nepraktické. Jako příklad konfigurace je uvedena část konfigurace Knot Resolveru běžícího na ODVR (Otevřené DNSSEC Validující Resolvery [19]).

Validace Lua, jako každý Turingovsky úplný jazyk, vede k tomu, že v případě konfigurace napsané v Lua musí být nejprve spuštěna v plném rozsahu, než ji je možné považovat za validní. Jednoduše řečeno, v případě použití Lua jazyku není možné konfiguraci validovat před tím, než je načtena do Knot Resolveru a spuštěna. To znamená, že překlepy v konfiguraci mohou například vést k syntakticky nesprávným doménovým jménům, které se odhalí až za běhu resolveru. Knot Resolver je tak také obtížné chránit před útočníkem, který provádí introspekci a přepisuje bezpečnostní omezení a hodnoty uvnitř programu. [2]

Použitelnost Nehledě na výběru programovacího jazyka pro konfiguraci, psaní takové konfigurace vyžaduje znalost syntaxe daného jazyka. Administrátoři nejsou programátoři a tak se pro ně stává konfigurace v jazyce Lua nečitelná, nepřehledná a hůře pochopitelná. Toto může vést k překlepům v konfiguraci, které projdou parserem, ale nefungují podle očekávání administrátora. [2]

Vyplývající požadavky:

- Deklarativní formát konfigurace pro Knot Resolver, který je jednoduše čitelný a použitelný i pro člověka.
- Validace konfigurace bez nutnosti načtení do Knot Resolveru a jeho spuštění.

Návrh řešení Deklarativní formátu konfigurace s omezenou sadou příkazů vycházející z YANG datového modelu a jazyka YAML, který je strojově zpracovatelný a zároveň dobře čitelný pro člověka. Navrhnout způsob konverze konfigurace mezi YAML a JSON formátem. Konfigurační data ve formátu JSON lze přímo validovat proti YANG datovému modelu nebo je importovat a exportovat z datového úložiště `sysrepo`.

5.2 Administrace různých implementací DNS resolveru

CZ.NIC, ale i další provozovatelé veřejných DNS resolverů provozují zároveň hned několik odlišných implementací, které potřebují stejnou nebo velmi podobnou konfiguraci. Diverzita síťového ekosystému podporuje odolnost služby vůči úplnému výpadku v případě, že by se u některé z implementací objevila kritická nebo jiná zranitelnost. [39]

Tato diverzita pochopitelně komplikuje administraci všech provozovaných DNS resolverů v síti. Konfigurační rozhraní, příkazy, konfigurační soubor a řízení nejsou mezi jednotlivými implementacemi DNS resolverů kompatibilní, například přenos konfigurace jedné implementace DNS resolveru na druhou je takřka nemožný. Operátoři musí vynaložit značné úsilí při administraci a často komplikovaně integrovat nějaký orchestrační nástroj pro automatizaci, například Ansible[6], Chef[7] nebo Puppet[8].

Vyplývající požadavky:

- Definice společné konfigurace, které je možné dosáhnout u různých implementací DNS resolveru.
- Jednotný mechanismus lokální i vzdálené administrace pro různé implementace DNS resoveru.

Návrh řešení Vytvoření YANG datového modelu (kapitola 3.3), který bude pokrývat společnou konfiguraci pro několik open-source implementací DNS resolveru. Tento datový model bude sloužit jako základ, který bude rozšířen datovým modelem pokrývající specifickou konfiguraci pro Knot Resolver. Jako mechanismus ke konfiguraci Knot Resolveru použít NETCONF protokol (kapitola 3), který je standardizovaným a dobře popsaným způsobem konfigurace síťových zařízení.

[6] `www.ansible.com`
[7] `www.chef.io`
[8] `www.puppet.com`

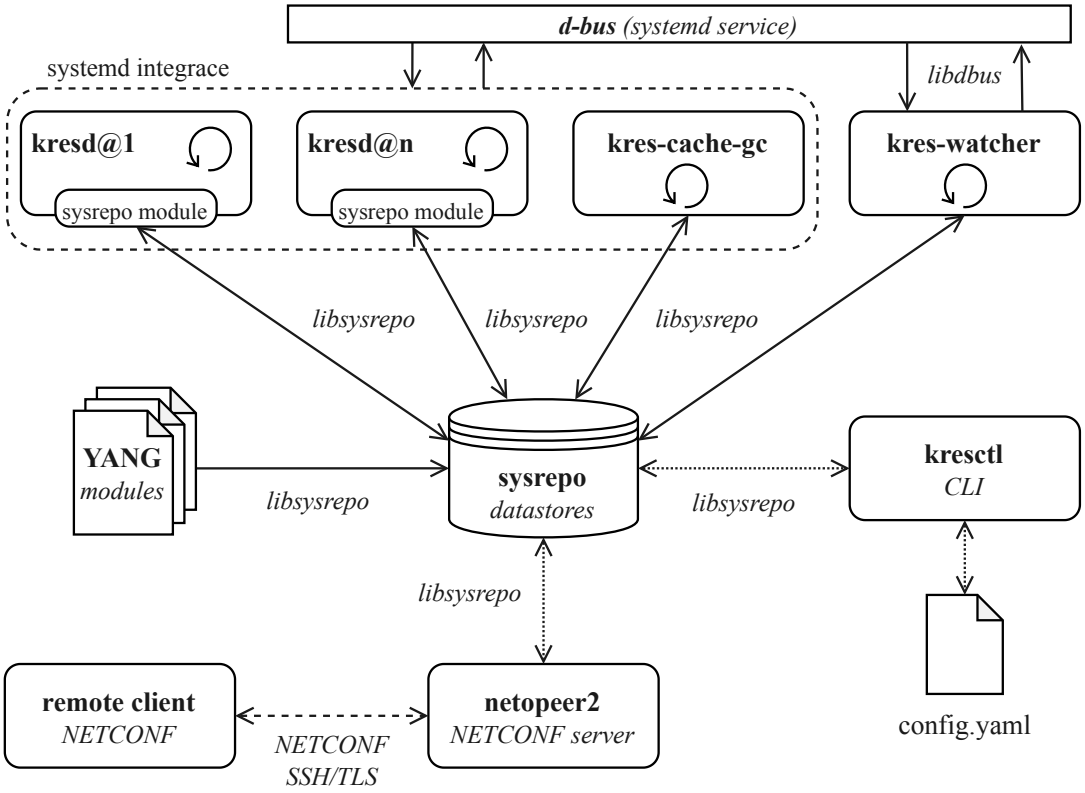
5.3 Dodatečné požadavky

Zde je výčet dílčích požadavků přímo od vývojového týmu Knot Resolveru. Tyto požadavky přímo neřeší problematiku konfigurace Knot Resolveru, ale je s nimi při návrhu systému potřeba počítat.

- Zachování současných přístupů ke konfiguraci z důvodů zpětné kompatibility a usnadnění případného přechodu ze starého způsobu konfigurace na novou.
- Perzistentní uchování konfigurace Knot Resolveru napříč restarty zařízení.
- Pokusit se navrhnout způsob centralizace operací, které jsou prováděné všemi instancemi `kresd` procesu. Tyto operace je reálně potřeba provádět pouze jednou.

6 Návrh systému

V této kapitole je popsán teoretický návrh systému pro řízení, konfiguraci a monitoring Knot Resolveru. Na následujícím obrázku je vizualizace celého systému.



Obrázek 6.1: Schéma navrhovaného systému

Jako centrální prvek pro správu konfiguračních a stavových dat jednotlivých procesů slouží datová úložiště (datastores) `sysrepo` (kapitola 4.1). Z tohoto úložiště si jednotlivé procesy odebírají svou konfiguraci, v případě požadavku poskytují svá stavová data a dostávají upozornění o změně konfigurace. K těmto účelů slouží knihovna *sysrepo*, pomocí které procesy komunikují s datovým úložištěm a provádějí potřebné operace. Knihovna tudíž musí být integrována do všech procesů, které mají jakoukoli potřebu komunikovat s úložištěm sysrepo. Každý proces se zaměřuje pouze na svou část konfigurace uloženou v `sysrepo` a definovanou datovým modelem (kapitola 8) v jazyce YANG. Knihovna `libyang` umožňuje konfigurační dat proti tomuto datovému modelu validovat.

K původním procesům `kresd` a `kres-cache-gc` přibývá nový proces pojmenovaný jako `kres-watcher`. Tento proces má na starosti řízení a monitoring ostatních procesů. Operace pro řízení a jejich parametry jsou stejně jako konfigurační a stavová data součástí datového modelu a je možné je volat pomocí knihovny `sysrepo`. Další důležitou úlohou tohoto procesu je provádění konfigurace vyžadující specifický přístup, tyto případy jsou popsány v kapitole 6.5.

Díky knihovně `sysrepo` jsou veškeré procesy Knot Resolveru napojené na úložiště `sysrepo`. Úprava konfigurace je tedy realizována změnou konfiguračních dat v datovém úložišti. Pro tento účel na lokální úrovni přímo slouží nové interaktivní administrační CLI (Command Line Interface) nazvané `kresctl`. Mimo konfiguraci slouží i k řízení a monitorování jednotlivých procesů. Implementace NETCONF serveru `netopeer2` a jeho přímé napojení na datové úložiště `sysrepo` umožňuje celému systému vzdálenou administraci pomocí NETCONF protokolu. Veškeré procesy tak lze vzdáleně konfigurovat pomocí NETCONF protokolu, díky RPC operacím je řídit a s pomocí notifikací je monitorovat.

6.1 Modul pro kresd

Integrace `sysrepo` do `kresd` procesu je provedena odděleně jako samostatný modul se stejným názvem. Zásluhou toho není nutné při realizaci návrhu zasahovat přímo do jádra `kresd` procesu a lze tak vyvíjet modul naprosto izolovaně. Cílem tohoto modulu je co nejefektivněji převádět konfiguraci, kterou získá z datového úložiště na konfiguraci Knot Resolveru, přesněji `kresd` procesu. Také by měl být schopen reagovat na žádosti o stavová data a v případě potřeby je poskytnout. Jelikož je možné modul implementovat v C nebo Lua, vyvstává otázka jaký jazyk bude pro implementaci vhodnější. Návrhem modulu pro konverzi konfigurace se tato práce nezaobírá. Na tomto modulu pracuje kolega a student MFF UK (Matematicko-fyzikální fakulta Univerzity Karlovy), s nímž máme dohodnuté aplikační rozhraní, které je reprezentováno YANG datovým modelem a knihovnou `sysrepo`.

6.2 kres-cache-gc

Proces `kres-chache-gc` se stará o uvolňování potřebného místa v cache paměti Knot Resolveru (kapitola 2.3.1). Tento proces se dá původně konfigurovat pouze přímo prostřednictvím parametrů při jeho spuštění. Napojení na datové úložiště `sysrepo` pomocí stejnojmenné knihovny mu umožňuje měnit konfigurační parametry i dynamicky za běhu. Navíc bude moci poskytovat stavová data a nebo případně posílat logy a statistiky v podobě asynchronních zpráv (YANG notifikace [1]). V YANG datovém modelu je pro `kres-chache-gc` proces určen uzel *garbage-collector* v kategorii `cache` (kapitola 8.3.3).

6.3 kres-watcher

Tento proces je nově vytvořený a stará se o řízení jednotlivých procesů `kresd` a `kres-cache-gc`, reaguje na řídící operace definované v datovém modelu jazykem YANG. Stará se také o speciální periodické operace a změny konfigurace (kapitola 6.5) cache paměti a instancí `kresd` procesu.

6.3.1 Řízení procesů

Procesy jsou automaticky řízeny na základě konfigurace datového modelu v sekci `server` (kapitola 8.3.1), kde je definováno jaké procesy a kolik jich má `kres-watcher` spustit. Procesy `kresd` a `kres-cache-gc` jsou řízeny prostřednictvím jejich dostupné `systemd` integrace přes softwarovou sběrnice `d-bus`. V případě potřeby manuálního řízení jednotlivých procesů jsou v datovém modelu definovány NETCONF RPC operace a akce (YANG actions), na které reaguje `kres-watcher` a provádí potřebné operace opět prostřednictvím `systemd` a `d-bus`. Tyto operace jsou detailněji popsány v kapitolách 8.2.2, 8.3.3 a 8.3.2. Úprava konfigurace a provádění operací, je možné pomocí konfiguračního rozhraní `kresctl` a nebo vzdáleně přes `netopeer2` server pomocí NETCONF klienta.

Stav procesů Mimo řízení procesů také poskytuje `kres-watcher` informace o stavu jednotlivých procesů, které získá s pomocí `systemd` a `D-Bus`. Stavová data o procesech je možné získat požadavkem na operační data procesů uložená v `sysrepo`. Pro stavová data procesů je v datovém modelu definován uzel `status` v kategorii `server` (kapitola 8.3.1) určený pouze ke čtení.

6.4 kresctl

Tento proces je jednoduché interaktivní terminálové rozhraní pro kompletní lokální administraci Knot Resolveru a jeho procesů. V jeho režii je také import a export konfiguračních dat ve formátu YAML a jeho konverze do formy zpracovatelné knihovnou `sysrepo`.

Pro přechod mezi jednotlivými stavy konfigurace (transakce) je využito datové úložiště `<candidate>` (kapitola 3.4). V tomto úložišti je změna konfigurace nejdříve připravena, poté validována proti datovému modelu a teprve po potvrzení je konfigurace převedena do `<running>` úložiště. Procesy Knot Resolveru dostanou upozornění konfigurace a změny aplikují.

6.4.1 Příkazy

Administrační příkazy pro `kresctl` vycházejí přímo z datového modelu (kapitola 8) pro Knot Resolver, kde jednotlivé uzly v datovém modelu jsou identifikovány pomocí jazyka XPath. Princip je ukázán na následujících příkladech, `drc` je prefix pro `cznic-resolver-common` modul a `kres` je prefix pro `cznic-resolver-knot` modul.

XPath	`drc:dns-resolver/cache/max-ttl`
Příkaz	`cache.max-ttl`

XPath	`drc:dns-resolver/server/kres:status`
Příkaz	`server.status`

XPath	`drc:dns-resolver/kres:instances[name='dot']/restart`
Příkaz	`instances.dot.restart`

Příkazy s parametry nastavují hodnoty konfiguračních dat v `sysrepo` a nebo volají RPC operace s parametry. Příkazy bez parametrů vypisují nastavené hodnoty, stavová data nebo volají RPC operace bez parametrů.

- `cache.max-ttl 86400`

  Nastaví hodnotu na 86400.

- `cache.max-ttl`

  Vypíše nastavenou hodnotu.

6.4.2 Konfigurační soubor

Návrh nového konfiguračního souboru spočívá ve využití JSON formátu konfiguračních dat, které je přímo možné importovat do `sysrepo` datových úložišť nebo v tomto formátu data z úložišť získat.

```
{
    "cznic-resolver-common:dns-resolver": {
        "cache": {
            "max-size": 104857600,
            "max-ttl": 172800,
            "min-ttl": 0,
            "cznic-resolver-knot:storage": "/etc/knot-resolver"
        }
    }
}
```

Zdrojový kód 6.1: JSON - ukázka konfiguračních dat

YAML Jako formát pro nový konfigurační soubor byl zvolen jazyk YAML, který je velmi dobře čitelný a přehledný i pro člověka. Jelikož jsou JSON a YAML formát částečně kompatibilní je možné je s drobnými omezeními mezi sebou konvertovat. Výsledný YAML formát je jednoduché oříznout o přebytečné názvy YANG modulů u jednotlivých uzlů a tím vytvořit velmi jednoduchý a přehledný konfigurační soubor.

```
cache:
    max-size: 104857600
    max-ttl: 172800
    min-ttl: 0
    storage: /etc/knot-resolver
```

Zdrojový kód 6.2: YAML - ukázka konfiguračních dat

Komentáře Při psaní konfigurace pomocí konfiguračních souborů je zvykem danou konfiguraci okomentovat aby bylo v budoucnu autorovi nebo jinému administrátorovi jasné co bylo konfigurací zamýšleno.

```
# This is example comment.
example: 104857600
```

Zdrojový kód 6.3: YAML - komentář konfiguračního souboru [37]

JSON bohužel na rozdíl od YAML nepodporuje komentáře. Pro zachování komentářů uvnitř datového úložiště `sysrepo` byly v datovém modelu definovány komentáře nově jako anotace (viz kapitola 8.3.5). Komentář v YAML formátu je tak možné konvertovat na JSON objekt, který se váže k některému z uzlů konfiguračních dat jako YANG metadata (RFC 7952 [37]).

```
{
    "@example": {
        "cznic-resolver-knot:comment":
            "This is example comment."
    },
    "example": 5000
{
```

Zdrojový kód 6.4: JSON - komentář definovaný YANG anotací [37]

6.5 Speciální případy konfigurace

Uvnitř Knot Resolveru je několik případů konfigurace a operací, které vyžadují odlišný přístup než ostatní (kapitola 5.1.1). Pro návrh možného řešení byly po konzultaci jako příklad vybrány dva případy, *cache prefill* a *TLS session ticket secret*, které jsou popsány v kapitolách níže.

6.5.1 Cache prefill

Jedná se o modul s názvem `prefill` [35] určený pro `kresd` proces. Tento modul umožňuje pravidelně předvyplnit cache paměť daty kořenové zóny (root zone) získaných přes HTTPS. Pokud tedy běží více instancí `kresd`, na každé z těchto instancí probíhá předvyplnění samostatně, i když mají všechny instance společnou cache paměť. Řešením je přesun `prefill` modulu na proces `kres-watcher`, který předvyplnění provede pouze jednou za požadovanou periodu a nebude provádět několikanásobně v závislosti na počtu `kresd` instancí.

6.5.2 TLS session ticket secret

Jde o parametr pro mechanizmus popsaný v RFC 5077 [49], který umožňuje TLS serveru obnovení relací a vyhnutí se udržování stavu relací na každého klienta. TLS server zapouzdří stav relace do tzv. *session ticket* a předá jej klientovi. Klient může následně pokračovat v relaci pomocí získaného tiketu.

Ke generování těchto tiketů slouží klíč, který je tvořen buďto náhodně nebo s použitím tajného řetězce (secret) prostřednictvím konfiguračního příkazu`tls_sticket_secret`. Pokud je tento tajný řetězec nakonfigurován stejný u všech instancí `kresd` procesu, obnovení relací klienta je možné provádět u jakékoli instance bez potřeby další komunikace mezi nimi. Pro bezpečnost musí mít také řetězec dostatek entropie, aby bylo těžké jej uhodnout. [35] Problém je řešen tím, že o konfiguraci `tls_sticket_secret` se bude starat opět proces `kres-watcher`. Bude periodicky generovat tajný řetězec (secret) s dostatečnou mírou entropie, který pokaždé uloží do datového úložiště `sysrepo`, odkud jej obdrží jednotlivé instance `kresd`.

7 Realizace systému

Tato kapitola se zaměřuje na samotnou implementaci systému pro administraci Knot Resolveru. Je zde popsána instalace a implementace jednotlivých knihoven, příprava datového úložiště `sysrepo`, implementace nových procesů a případné řešené problémy. Systém je kromě datového modelu (kapitola 8) realizovaný v jazyce C nebo případně Lua. Je důležité zmínit, že některé části zdrojového kódu představované v této kapitole byly upraveny, aby z nich bylo možné pochopit kontext. Proto se může lišit kód uvedený zde a přímo ve zdrojových souborech.

7.1 Příprava

Před začátkem realizace navrženého systému je potřeba nainstalovat některé knihovny. Dále, aby bylo možné během implementace a testování získat data, musí být do datového úložiště `sysrepo` importována konfigurační data.

7.1.1 Instalace knihoven

Pro integraci `sysrepo` datového úložiště a knihovny je použita `devel` větev z github repositáře projektu. Hlavní závislostí pro `sysrepo` je knihovna `libyang` taktéž v `devel` větvi a některé další balíčky, které jsou běžnou součástí distribucí Linuxu. [15]

- C compiler (gcc >= 4.8.4, clang >= 3.0, ..)
- cmake >= 2.8.12
- libpcre (devel)
- cmocka >= 1.0.0 (pouze pro testy)

```
$ git clone https://github.com/CESNET/libyang.git
$ cd libyang && git checkout devel
$ mkdir build && cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
$ make
# make install
```

Zdrojový kód 7.1: Instalace knihovny libyang [15]

Pro úspěšnou instalaci `sysrepo` je také důležité určit, kde se bude nacházet `sysrepo` repositář `-DREPO_PATH`, který bude obsahovat startovní konfigurační data a nainstalované YANG moduly. Testy pro potvrzení úspěšné instalace je možné spustit pomocí příkazu `make test`.

```
$ git clone https://github.com/CESNET/sysrepo.git
$ cd sysrepo && git checkout devel
$ mkdir build && cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr \
        -DREPO_PATH=/home/alesmrazek/repository ..
$ make
# make install
```

Zdrojový kód 7.2: Instalace sysrepo knihovny a datového úložiště [51]

7.1.2 Instalace datového modelu

Datový model, veškeré potřebné YANG moduly a příklad konfiguračních dat je přiložen ve složce `yang-model` (viz. přílohy). YANG moduly je možné do `sysrepo` instalovat pomocí nástroje `sysrepoctl`. Stačí přejít do složky s YANG moduly a instalovat modul `cznic-resolver.knot.yang`. Ostatní potřebné moduly včetně `cznic-resolverl-common.yang` jsou ze složky importovány nebo instalovány automaticky. Pomocí příkazu `sysrepoctl -l` je poté možné vypsat seznam všech instalovaných a importovaných modulů v `sysrepo`.

```
$ cd yang-modules
$ sysrepoctl -i cznic-resolver-knot.yang -s .
$ sysrepoctl -l
```

Po úspěšně nainstalovaném datovém modelu je potřeba naplnit úložiště daty, aby bylo možné systém testovat. Prvotní naplnění datového úložiště konfiguračními daty je možné provést pomocí nástroje `sysrepocfg`. Načtením konfiguračních dat do `<startup>` datového úložiště je zajištěna perzistentní konfigurace napříč restarty zařízení, tedy není potřeba po restartu systému opakovaně data do úložiště vkládat.

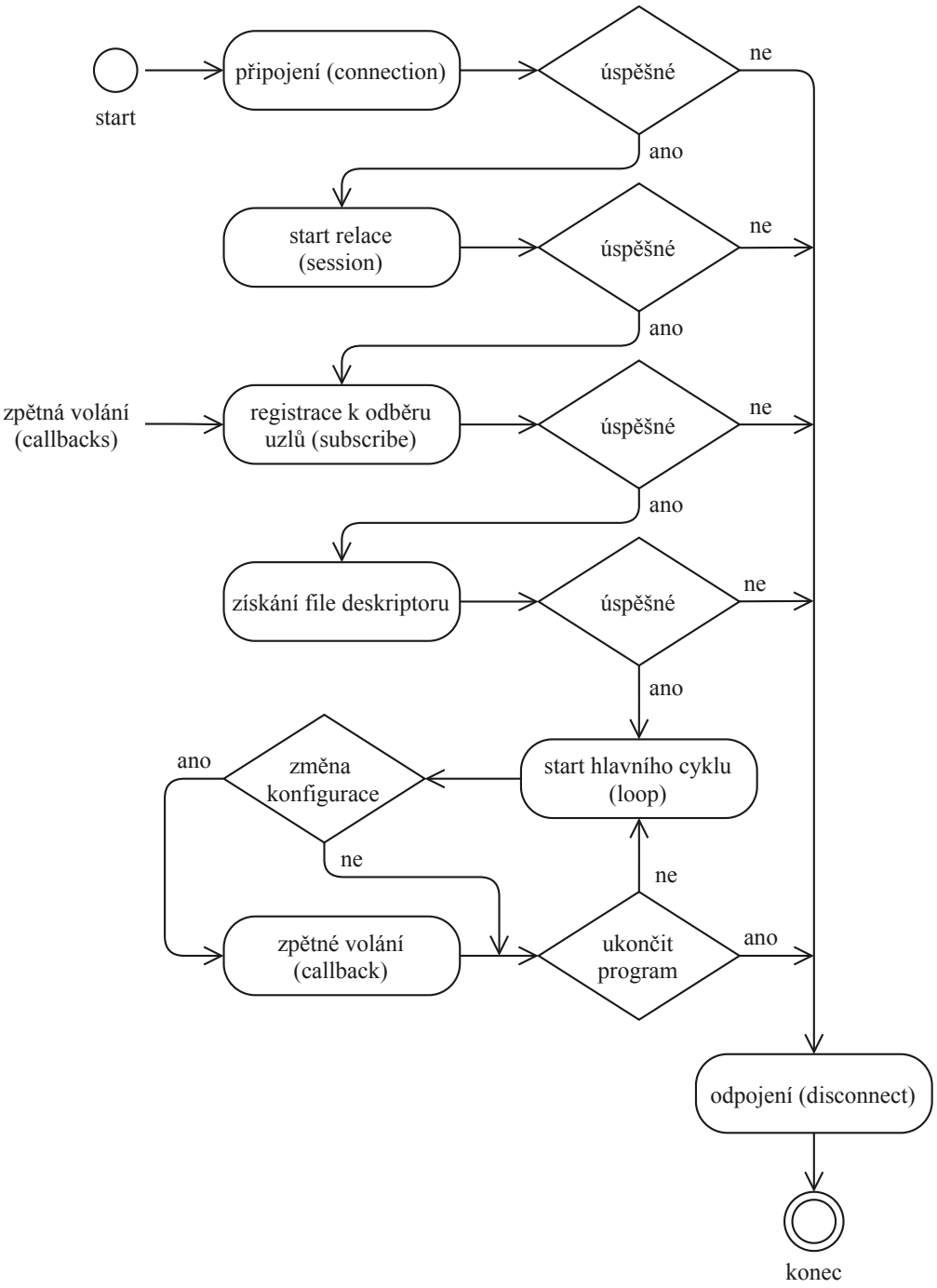
```
$ cd yang-modules
$ sysrepocfg --import=data/example-startup-config.json --datastore startup
 ↪  --module cznic-resolver-common
```

7.2 sysrepo integrace

Aby mohl proces průběžně odebírat konfiguraci a dostával upozornění o změnách konfigurace, musí po celou dobu provozu držet spojení s datovým úložištěm pomocí knihovny `sysrepo`. Integrace `sysrepo` s procesy Knot Resolveru lze docílit několika způsoby. V zásadě se dělí na přímé a nepřímé způsoby, kdy je `sysrepo` integrováno přímo v kódu procesu a nebo se o změny v konfiguraci stará samostatný proces. Jelikož jsou jednotlivé procesy Knot Resolveru jedno-vláknové, bylo pro zachování architektury `sysrepo` integrováno přímím způsobem s použitím file deskriptoru.

7.2.1 Životní cyklus sysrepo připojení

V této kapitole je popsán způsob integrace jakým proces odebírá konfiguraci z datového úložiště. Na následujícím diagramu je naznačen průběh připojení procesu k `sysrepo` a k odběru změn v konfiguraci.



Obrázek 7.1: Životní cyklu sysrepo připojení

Kontext přípojení k `sysrepo` je reprezentovaný dvěma proměnnými. První reprezentuje připojení a druhá reprezentuje relaci (session) s datovým úložištěm. Jednoduše řečeno, relace (session) určuje datové úložiště, nad kterým budou následné operace prováděny.

```
sr_conn_ctx_t *connection = NULL;
sr_session_ctx_t *session = NULL;

int ret = sr_connect(0, &connection);

/* Vytvoření relace s running datovým úložištěm. */
if (!ret) ret = sr_session_start(connection, SR_DS_RUNNING, &session);
```

Zdrojový kód 7.3: Inicializace sysrepo připojení

Po vytvoření relace s datovým úložištěm se může proces přihlásit k odběru konfiguračních dat na určitém uzlu a jako parametr vložit funkci, která bude při změně konfigurace zpětně volána. Parametr `SR_SUBSCR_NO_THREAD` znamená, že nebude vytvořeno samostatné vlákno pro zpracování u tohoto odběru uzlu a následně může proces získat file deskriptor. Záleží na implementaci hlavního cyklu programu, kdy je v případě události na file deskriptoru volána funkce `sr_process_events()`, která vynutí případná zpětná volání funkcí registrovaných při odběru (callbacks). Další důležitý parametr je `SR_SUBSCR_ENABLED`, který způsobuje vynucení zpětného volání (callback) okamžitě v momentě, kdy se proces přihlásí k odběru uzlu, toho je využíváno při startu procesů k rychlému získání aktuální konfigurace. Pro přihlášení k odběru stavových dat a RPC operací slouží velmi podobně funkce `sr_oper_get_items_subscribe()` a `sr_rpc_subscribe()`.

```
#define YM_COMMON   "cznic-resolver-common"
#define XPATH_BASE  "/"YM_COMMON":dns-resolver"

int fd;
struct pollfd fds[1];
sr_subscription_ctx_t *subscription = NULL;

/* Odběr konfiguračních dat v kategorii server */
ret = sr_module_change_subscribe(session, YM_COMMON, XPATH_BASE"/server/*",
 ↪  server_change_cb, NULL, 0, SR_SUBSCR_NO_THREAD|SR_SUBSCR_ENABLED,
 ↪  &sr_subscription);

/* Získání file descriptoru */
ret = sr_get_event_pipe(sr_subscription, &fd);
```

Zdrojový kód 7.4: Inicializace sysrepo odběru uzlů

Když nastane změna v konfiguraci, ke které je proces přihlášen k odběru, je zpětně volána funkce, která je zadána při registraci odběru k danému uzlu. V tomto případě se jedná o `server_change_cb` reagující na změny konfigurace v podstromu `server`. Proces aplikace změny konfigurace má dvě fáze, v té první je zavolána funkce s parametrem události `SR_EV_CHANGE`, která procesu říká, že nastala změna konfigurace a zda ji chce aplikovat. Zde může proběhnout například validace. Pokud proces vrátí funkcí kladnou odpověď, je následně funkce zavolána znovu s `SR_EV_DONE` a konfigurace je aplikována. V případě negativní odpovědi je zavolána funkce s událostí `SR_EV_ABORT` kdy někde nastala chyba a konfigurace nemůže být aplikována. Podobně jsou volány zpětně funkce na požadavek stavových dat a volání RPC operací s rozdílem, že jsou volány pouze jednou.

```
static int server_change_cb(sr_session_ctx_t *session, const char
 ↪  *module_name, const char *xpath, sr_event_t event, uint32_t request_id,
 ↪  void *private_data)
{
        if(event == SR_EV_CHANGE)
        {
                /* validace konfigurace*/
        }
        else if (event == SR_EV_DONE)
        {
                /* aplikace konfigurace */
        }
        else if(event == SR_EV_ABORT)
        {
                /* přerušení */
        }
        return SR_ERR_OK;
}
```

Zdrojový kód 7.5: Zpětné volání (callback) pro odběr konfiguračních dat

7.2.2 Logování a monitoring

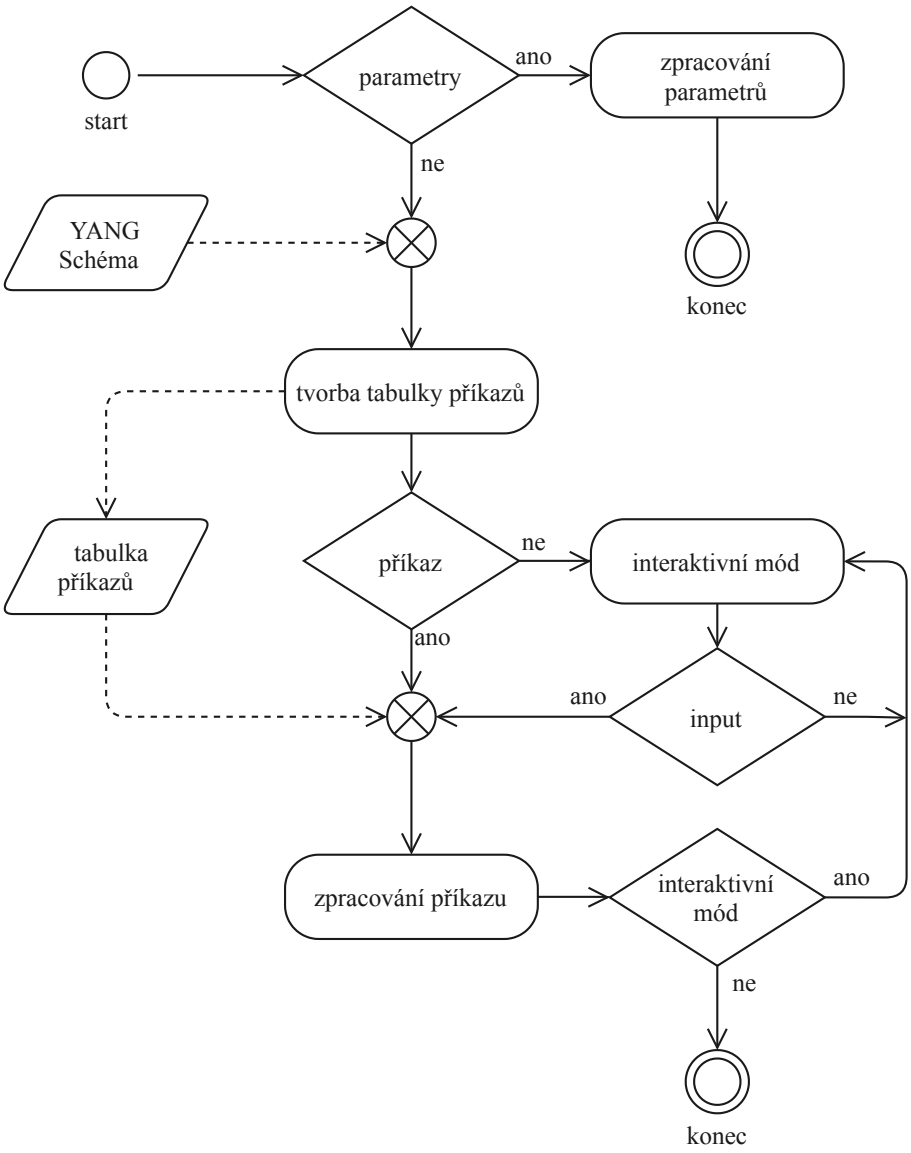
Pro monitoring a logování je v datovém modelu definován uzel pro notifikace (kapitola 8.3.4). Zprávy jsou na tento uzel posílány pomocí `sysrepo` funkce `sr_event_notif_send_tree()`. Následně poslanou zprávu obdrží všichni odběratelé uzlu na který je zpráva odeslána, například příkaz rozhraní `kresctl logging.log` začne odposlouchávat tyto zprávy.

```c
time_t seconds;
struct lyd_node *notif = NULL;

seconds = time(NULL);

/* Vytvoření uzlu pro notifikaci */
notif = lyd_new_path(NULL, ctx, path, NULL, 0, 0);

/* Doplnění vstupních parametrů notifikace */
lyd_new_path(notif, NULL, "log-type", "info", 0, 0)
lyd_new_path(notif, NULL, "time-stamp", seconds, 0, 0)
lyd_new_path(notif, NULL, "process", "kres-watcher", 0, 0)
lyd_new_path(notif, NULL, "message", "new tls sticket secret generated", 0, 0)

/* Odeslání zprávy */
ret = sr_event_notif_send_tree(session, notif);
```

Zdrojový kód 7.6: C - Logování pomocí sysrepo NETCONF notifikací

7.3 kresctl

Základem pro tvorbu tohoto administračního rozhraní je použita knihovna `histedit` starající se o interaktivní prostředí a historii zadaných příkazů. `kresctl` je možné spustit ve dvou režimech, buďto s příkazem přímo z příkazové řádky nebo interaktivně pro zadání série příkazů. Specialitou je, že administrační příkazy jsou tvořeny až při spuštění přímo z datového modelu.



Obrázek 7.2: Vývojový diagram - administrační rozhraní kresctl

Administrační rozhraní `kresctl` podporuje dva režimy práce:

- Pokud je `kresctl` spuštěno s parametry nebo příkazy, je na základě příkazu nebo parametru provedena operace a následně je proces ukončen.
- Spuštění `kresctl` bez parametrů a dalších příkazů spustí interaktivní konfigurační rozhraní, které začíná řádkem označeným `kresctl>`. Pomocí tabulátoru nebo příkazu `help` je pak možné vypsat nápovědu.

Konfigurační rozhraní má zabudované pouze základní příkazy a zbylé příkazy jsou tvořeny až při jeho startu z datového modelu. Některé zabudované příkazy se starají o transakci nad `candidate` datovým úložištěm pro přípravu konfigurace a její následné přesunutí (commit) do `running` úložiště. Další slouží pro import a export konfigurace v souboru.

```
$ kresctl --help
Usage:
 kresctl [parameters] <command> [command_arguments]

Parameters:
 -h, --help               Print the program help.
 -V, --version            Print the program version.

Commands:
 exit                     Exit the program.
 help                     Print the program help.
 version                  Print the program version.

 import     <file-path>   Import YAML configuration file.
 export     <file-path>   Export YAML configuration file.

 begin                    Begin a transaction.
 commit                   Commit a transaction.
 abort                    Abort a transaction.
 validate                 Validate a transaction changes.
 diff                     Show configuration changes.
```

Zdrojový kód 7.7: Sada zabudovaných příkazů v rozhraní kresctl

7.3.1 Tvoření příkazů

Tvorba příkazů je realizována pomocí `sysrepo` funkce `sr_get_context()`. Tato funkce umožňuje získat `libyang` kontext používaný konkrétním připojením k `sysrepo` a může být použit pro získání kořenového uzlu schématu datového modelu. Následně je možné procházet celým schématem funkcí `lys_getnext` a při tom vytvářet příkazy a ukládat je do tabulky. Tvoření textové podoby příkazů vychází z kapitoly 6.4.1.

```
ly_context = sr_get_context(connection);
root_node = ly_ctx_get_node(ly_context, NULL,
↪  "/cznic-resolver-common:dns-resolver", 0);

next_node = lys_getnext(NULL, root_node, NULL, 0);
```

Zdrojový kód 7.8: C - získání YANG schémata datového modelu

Příkazy pak v nápovědě `kresctl` vypadají následovně.

```
cache                          Parameters of the resolver cache.
cache.current-size             Current size of the cache.
cache.max-size      <uint64>   Maximum size of the cache.
cache.min-ttl       <uint32>   Minimum time-to-live for cache entries.
cache.max-ttl       <uint32>   Maximum time-to-live for cache entries.
cache.storage       <fs-path>  Knot Resolver cache storage.
```

Zdrojový kód 7.9: Ukázka příkazů vytvářených až při startu kresctl.

Konfigurační soubor Ke konverzi mezi JSON a YAML formátem konfigurace byly použity knihovny `LibYAML` a `YAJL`.

7.4 kres-watcher

Pro realizaci tohoto nového procesu byla zvolena knihovna `libuv`[9], kterou používá i proces `kresd` pro realizaci hlavní smyčky programu zvanou jako event loop. Tento způsob realizace umožňuje jednoduché rozšiřování a přidávání nových funkcionalit do hlavní smyčky programu.

7.4.1 Řízení procesů

Zbylé procesy Knot Resolveru jsou řízeny na základě konfigurace dostupné v kategorii `server` (kapitola 8.3.1) datového modelu a nebo manuálně pomocí RPC operací (kapitola 8.2.2).

[9] https://www.libuv.org

Pro implementaci řízení jednotlivých procesů `kresd` a `kres-cache-gc` je použita knihovna `sd-bus`. Knihovna je přímo součástí `systemd` [53] a ulehčuje řízení integrovaných služeb pomocí softwarové sběrnice `D-Bus`. Nutno podotknout, že tato knihovna není vůbec dobře zdokumentována, a přijít na způsob řízení s její pomocí nebylo jednoduché. Knihovna ale byla vybrána z důvodu, že v dokumentaci `D-Bus` varují přímo před použitím jejich nízkoúrovňového C API[10].

```
sd_bus *bus = NULL;
sd_bus_message *reply = NULL;
sd_bus_error error = SD_BUS_ERROR_NULL;

int ret = sd_bus_default_system(&bus);

ret = sd_bus_call_method(bus,
    "org.freedesktop.systemd1",
    "/org/freedesktop/systemd1",
    "org.freedesktop.systemd1.Manager",
    "StartUnit",
    &error,
    &reply,
    "ss",
    "kres-cache-gc.service",
    "replace"
);
```

Zdrojový kód 7.10: C - řízení procesu prostřednictvím knihovny sd-bus

Stav jednotlivých procesů je možné zjistit dotazem na hodnotu vlastnosti `ActiveState` v jejich `D-Bus` rozhraní, které je při jejich spuštění pomocí `systemd` automaticky vytvořeno. Zjišťování stavu probíhá až v momentě kdy je zaslána žádost o stavová data uživatelem.

[10] https://dbus.freedesktop.org/doc/api/html/

```c
char *status;
sd_bus *bus = NULL;
sd_bus_error error = SD_BUS_ERROR_NULL;

int ret = sd_bus_default_system(&bus);

ret = sd_bus_get_property_string(
        bus,
        "org.freedesktop.systemd1",
        "org.freedesktop.systemd1.Unit",
        "/org.freedesktop.systemd1/unit/kres_2dcache_2dgc_2eservice",
        "ActiveState",
        &error,
        &status
);
```

Zdrojový kód 7.11: C - získání stavu procesu prostřednictvím knihovny sd-bus

7.4.2 TLS session ticket secret

Jedná se o jeden z případů konfigurace vyžadující speciální přístup (kapitola 6.5.2). Periodické generování tajného řetězce (secret), který je následně nastaven v *sysrepo* odkud jej odebírají veškeré spuštěné instance `kresd` procesu.

Jako časovač (timer) pravidelné změny tajného řetězce je použity funkce `uv_timer` knihovny `libuv`. V případě, že je hodnota v `sysrepo` nastavena ručně uživatelem, `kres-watcher` odebírá změny konfigurace tohoto uzlu a ve zpětném volání (callback) pouze aktualizuje časovač (timer) a po jeho vypršení vygeneruje nový tajný řetězec. Pro generování náhodného řetězce byla použita knihovna `GnuTLS` a její funkce `gnutls_session_ticket_key_generate()`. Výsledný řetězec je převeden do `BASE64` a pomocí `sysrepo` knihovny je hodnota nastavena v datovém úložišti.

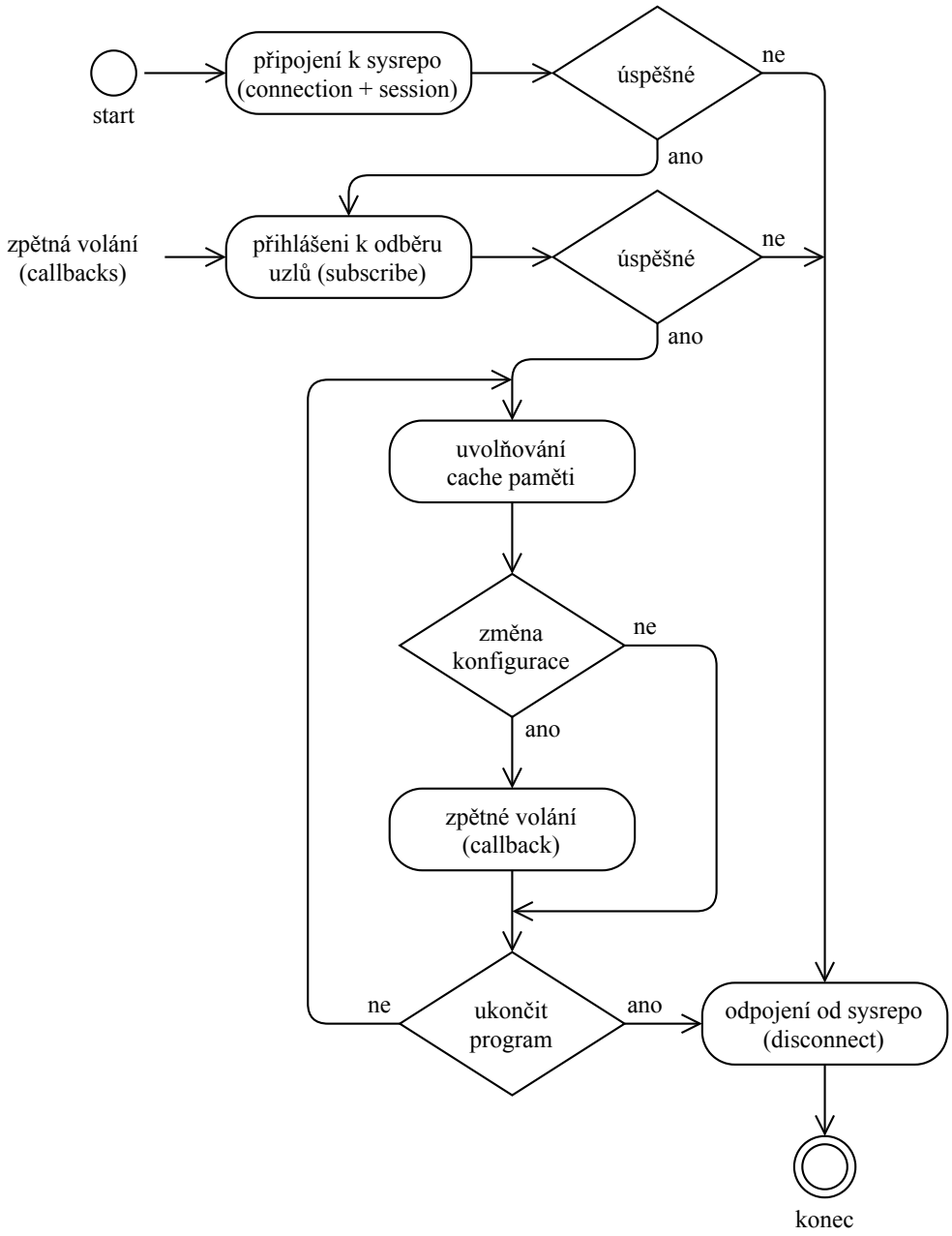
Při nastavování hodnoty do datového úložiště `sysrepo` vznikal problém. `kres-watcher` totiž zároveň uzel konfigurující tajný řetězec odebírá aby mohl nastavit časovač (timer) pro jeho opětovné generování a změnu v konfiguraci. Při pokusu o nastavení uzlu tedy vznikal deadlock, jelikož `kres-watcher` při změně konfigurace čekal na potvrzení že odběratelé uzlu konfiguraci přijali a `kres-watcher` byl zároveň procesem odebírajícím konfiguraci na tomto uzlu. Problém byl nakonec vyřešen parametrem `SR_SUBSCR_DONE_ONLY` při registraci odběru uzlu. Proces dostane upozornění až po aplikování konfigurace ostatními odebírajícími procesy a nedostane možnost konfiguraci verifikovat. Toto ale nevadí jelikož `kres-watcher` konfiguraci neaplikuje a tak ověření mohou provést ostatní procesy.

7.4.3 Cache prefill

`prefill` je modul pro `kresd` proces, který periodicky předvyplňuje cache paměť (kapitola 6.5.1). Aby bylo možné tento modul načíst i do `kre-watcher`, byl tento proces postaven na základech zdrojovém kódu `kresd` procesu, který podporuje načítání modulů a byly vypnuty nepotřebné funkce. Tedy `kres-watcher` se přihlašuje k odběru konfigurace `prefill` modulu v sekci `cache`. Pokud je v konfiguračních datech přítomna konfigurace pro tento modul, je automaticky načten a nakonfigurován.

7.5 kres-cache-gc

Při spuštění vytvoří `kres-cache-gc` proces spojení s datovým úložištěm `sysrepo` a zaregistruje se k odběru uzlů obsahující jeho konfiguraci a stavová data (kapitola 8.3.3). Mimo svou vlastní konfiguraci odebírá také uzel `storage` v sekci `cache`, aby věděl, kde má cache paměť při jejím vyprazdňování hledat. Při registraci odběrů uzlů okamžitě obdrží současnou konfiguraci uloženou v `sysrepo` a uloží ji do svých vnitřních struktur. Poté je spuštěn hlavní cyklus programu, ve které je kontrolováno zda nedošlo ke změně konfigurace nebo neproběhla žádost o stavová data. Při ukončení programu se zruší spojení se `sysrepo` a odběr všech uzlů.



Obrázek 7.3: Vývojový diagram - kres-cache-gc s integrací sysrepo

Integrace `sysrepo` do procesu `kres-cache-gc` je velmi jednoduchá. Tento proces je původně realizovaný jednoduchým `do-while` cyklem, ve kterém je uspán pomocí funkce `usleep()` na dobu určenou intervalem vyprazdňování cache paměti. Jelikož `kres-cache-gc` proces komunikuje se `sysrepo` pomocí file deskriptoru, byla funkce `usleep()` nahrazena funkcí `poll()`. Tato funkce umožňuje čekat na události na file deskriptoru, tedy upozornění o změně konfiguračních dat nebo žádost o data stavová, a zároveň uspat proces na dobu intervalu kontroly cache paměti. Tím pádem je nově možné konfigurovat `kres-cache-gc` i za běhu.

Požadavkem od vývojového týmu Knot Resolveru byla možnost kompilovat tento proces i bez knihovny `sysrepo`, tedy v původním stavu před integrací `sysrepo`. Toho bylo docíleno pomocí `#ifdef` podmínky na části kódu se `sysrepo` integrací a v případě nedostupnosti `sysrepo` knihovny se zkompiluje původní část kódu.

```
#ifdef ENABLE_SYSREPO
        int poll_res = poll(fds, 1, (int)cfg.gc_interval/1000);
        if(poll_res > 0)
                sr_process_events(sr_subscription, sr_session, NULL);
        else if (poll_res < 0){
                ret = errno;
                if (rv && rv != EINTR)
                        printf("Error (%s)\n", strerror(rv));
                goto cleanup;
        }
#else
        usleep(cfg.gc_interval);
#endif
```

Zdrojový kód 7.12: Integrace sysrepo v kres-cache-gc procesu

7.6 Kompilace a instalace Knot Resolveru

Knot Resolver využívá `meson` build systém [54] a po každé změně kódu musí být znovu zkompilován. Pro kompilaci a instalaci ze zdrojového kódu Knot Resolver vyžaduje relativně velké množství závislostí. Výčet všech závislostí je popsán v dokumentaci Knot Resolveru v sekci *Building from sources* [35]. Při vývoji byl také použit při kompilaci parametr `-Db_sanitize=address`, který umožňuje ověřit stav dynamicky alokované paměti při běhu procesů.

```
$ meson build_dir --prefix=/tmp/kr --default-library=static
$ meson build -Db_sanitize=address --prefix=/tmp/kr
↪  --default-library=static
$ ninja -C build_dir
$ ninja install -C build_dir
```

Zdrojový kód 7.13: Kompilace Knot Resolveru [35]

7.7 Spuštění

Jelikož je `kres-watcher` novým centrálním prvkem systému, spuštění Knot Resolveru je realizováno spuštěním *kres-watcher* procesu. Pro jednoduchost je služba pro `systemd` integraci pojmenována jako `knot-resolver`.

```
$ sudo systemctl start knot-resolver
```

Nebo může být `kres-watcher` zapnutý napříč restarty systému.

```
$ sudo systemctl enable knot-resolver
```

Příkaz spustí pomocí `systemd` integrace proces `kres-watcher`, který následně na základě konfigurace v `sysrepo` spustí všechny další potřebné procesy opět přes `systemd` rozhraní. Spuštěné procesy nakonec aplikují svou konfiguraci ze `sysrepo`. Stav jednotlivých procesů Knot Resolveru je možné ověřit pomocí administračního rozhraní `kresctl`.

```
$ kresctl server.status
```

Pokud například na základě konfigurace není spuštěn proces `kres-cache-gc`, může být spuštěn následujícím příkazem.

```
$ kresctl cache.garbage-collector.start
```

8 Datový model

Datový model pro administraci Knot Resolveru a jeho procesů je definovaný pomocí modelovacího jazyka YANG (kapitola 3.3). Tento datový model neurčuje pouze strukturu konfiguračních a stavových dat, ale také definuje operace pro řízení a monitoring. Datový model vytvořený v rámci práce nepokrývá kompletní konfiguraci Knot Resolveru, ale zaměřuje se na vybranou sadu, která má prověřit možnosti a funkčnost navrženého systému. Tato konfigurační sada vychází ze základních konfiguračních parametrů DNS resolverů. Mimo nekompletní konfiguraci by měl datový model pokrýt veškeré důležité administrační operace pro řízení a monitorování. Konfigurační sada je dále po debatě s konzultantem rozšířena o části konfigurace vyžadující ze strany procesů speciální přístup při jejich návrhu a implementaci (kapitola 6.5).

Při návrhu datového modelu je důležité si uvědomit, že datový model má zásadní vliv na podobu výsledného systém, jelikož na něm stojí podoba příkazů a konfiguračního souboru pro `kresctl` (kapitola 6.4) a také integrace knihovny `sysrepo` v jednotlivých procesech. Kompletní navržený model reprezentovaný jako stromové schéma (YANG Tree Diagram [12]) je vizualizován v příloze A. Příklad konfiguračních dat ve formátu JSON a YAML je možné vidět v přílohách C a D.

8.1 YANG moduly

Datový model pro Knot Resolver je definován dvěma vytvořenými YANG moduly, jedním základním a druhým rozšiřujícím. Základní modul nazvaný `cznic-resolver-common` (kapitola 8.2) má představovat průnik konfiguracemi několika vybraných open-source implementací DNS resolveru. Druhý modul `cznic-resolver-knot` (kapitola 8.3) tento základní modul rozšiřuje a definuje konfiguraci specifickou pro Knot Resolver a jeho procesy. Mimo tyto YANG moduly jsou použity některé již existující a specifikované moduly. Takovéto moduly poskytují definici nových datových typů nebo dodatečná rozšíření pro jazyk YANG, která nejsou součástí základní specifikace. Vytvořené YANG moduly jsou popsány ve dvou následujících kapitolách, ostatní použité YANG moduly jsou stručně představeny v následujícím výčtu.

- **ietf-yang-metadata@2016-08-05**

  Tento modul zavádí rozšíření, které umožňuje definovat anotace jako metadata k jednotlivým uzlům ve schématu. RFC 7952 [37]

- **ietf-netconf-notifications@2012-02-06**

  Tento modul definuje datový model, který umožňuje NETCONF klientovi obdržet notifikace o základní událostech NETCONF protokolu. RFC 5277 [1]

- **iana-dns-class-rr-type@2018-10-26**

  Tento modul převádí IANA registry DNS CLASS a Resource Record (RR) typů na typy (typedef) odvozené od jazyku YANG.

- **cznic-dns-rdata@2018-12-10**

  Tento modul zavádí nové datové typy (typedef) a shlukování (grouping), které definují obsah RDATA pro všechny typy záznamů DNS (RR, resource records).

Veškeré použité YANG moduly jsou přiloženy v elektronické podobě, viz přílohy.

8.2 Společný YANG modul pro DNS resolvery

Základní datový model konfigurace pro DNS resolvery je definován YANG modulem `cznic-resolvers-common`, který reaguje na problematiku popsanou v kapitole 5.2. Tento modul představuje průnik konfigurace open-source implementacemi DNS resolveru Unbound [55], PowerDNS Recursor [47] a Knot Resolver.

Záměrem tohoto datového modelu je umožnit konfigurovat vybrané implementace DNS resolveru prostřednictvím NETCONF protokolu. Na pozadí této diplomové práce se totiž skrývá myšlenka administrace současně několika rozdílných implementací DNS resolveru pomocí jednoho datového modelu. Provozovatelé síti totiž z důvodů bezpečnosti používají většinou nejméně dvě rozdílné implementace DNS resolverů, které je potřeba nakonfigurovat shodně. Možnost konfigurovat tyto rozdílné implementace současně by značně ulehčilo jejich administraci a konfigurace mezi nimi by byla částečně přenositelná.

Zásadním problémem myšlenky společného modelu je, že i když jednotlivé implementace potřebují konfigurovat velmi podobné parametry, každá implementace DNS resolveru používá naprosto odlišnou syntaxi, rozsah nastavení nebo výchozí hodnoty pro konfiguraci parametru. Proto bylo potřeba udělat detailní průzkum konfigurace těchto implementací, kde byl hledán optimální model pro konfiguraci mezi jednotlivými implementacemi (tabulka 8.1). Kompletní tabulka porovnání konfigurace jednotlivých implementací je dostupná v elektronické příloze, viz přílohy. Datový model reprezentuje pouze strukturu konfiguračních a stavových dat, případně RPC operace pro řízení nebo notifikace pro monitorování. Samozřejmě každá implementace DNS resolveru má i své specifické části konfigurace, které nemohou být zahrnuty do tohoto datového modelu. Tyto specifické části jsou v případě Knot Resolveru do celkového datového modelu zahrnuty prostřednictvím rozšiřujícího YANG modulu (kapitola 8.3). Aplikování konfigurace je ponecháno na konkrétní implementaci DNS resolveru, jako je například v této práci navrhovaný systém pro administraci Knot Resolveru (kapitola 6).

Knot Resolver	**Unbound**	**PowerDNS Recursor**
net = {*ip_addr@port, ...*}	interface: *ip_addr@port*	local-address=*ip_addr*:*port*
net.outgoing_v4(*ipv4_addr*)	outgoing-interface: *ipv4_addr*	query-local-address=*ipv4_addr*
net.outgoing_v6(*ipv6_addr*)	outgoing-interface: *ipv6_addr*	query-local-address6=*ipv6_addr*
cache.max_ttl(*172000*)	cache-max-ttl: *172000*	max-cache-ttl=*172000*
verbose(*true/false*)	verbosity: *0-5*	loglevel=*0-9*

Tabulka 8.1: Příklad porovnání konfigurace implementací DNS resolveru. [47, 35, 55]

8.2.1 Základní kategorie

Datový model je na první úrovni větven do kategorií, které vycházejících z logického členění konfigurace podle její povahy. Kategorie se tak mohou dále jednoduše větvit nebo rozšiřovat podle potřeby. Základní kategorie datového modelu jsou popsány v následujícím seznamu.

- **server**

  Obecná nastavení pro DNS resolver server, jedná se o parametry ovlivňující podobu spuštění programu reprezentujícího DNS resolver na serveru.

- **network**

  Konfigurace týkající se síťového připojení, konfigurace protokolů, portů nebo TLS a HTTP komunikace.

- **resolver**

  Parametry ovlivňující přímo činnost DNS resolveru, doménové zóny, jmenné servery a další nastavení týkající se přímo zpracování DNS dotazů.

- **dnssec**

  Parametry pro zabezpečení pomocí funkcionality DNSSEC (kapitola 1.8), zahrnuje též seznamy důvěryhodných a nedůvěryhodných zdrojů.

- **cache**

  Veškeré parametry ovlivňující fungování cache paměti, například umístění, velikost nebo TTL.

- **dns64**

  Nastavení parametrů pro DNS64, IPv6 prefix (kapitola 1.9).

- **logging**

  Konfigurační parametry související s logováním, například úroveň logování atp..

Další členění datového modelu je znázorněno v YANG schématu (YANG Tree Diagram [12]) v příloze A. V následujících kapitolách jsou detailně vysvětleny některé části datového modelu.

8.2.2 Operace pro řízení

Pro řízení DNS resolveru pomocí pomocí NETCONF protokolu jsou v datovém modelu definovány RPC (Remote Procedure Call) operace definující řídící operace. V případě navrhovaného systému pro Knot Resolver na tyto operace reaguje proces `kres-watcher` (kapitola 6.3], který spustí, restartuje nebo vypne veškeré potřebné procesy.

```
module: cznic-resolver-common
   rpcs:
      +---x start
      |  +--ro input
      |  +--ro output
      +---x stop
      |  +--ro input
      |  +--ro output
      +---x restart
         +--ro input
         +--ro output
```

Zdrojový kód 8.1: YANG Tree Diagram - RPC operace určené k řízení DNS resolveru

8.3 Rozšiřující YANG modul pro Knot Resolver

Jestliže společný datový model pro DNS resolvery 8.2 (kapitola 8.2) určuje společné konfigurační parametry napříč implementacemi DNS resolverů, tak rozšiřující YANG modul `cznic-resolver-knot` určuje datový model specifický pouze pro Knot Resolver. Úkolem datového modelu definovaného tímto modulem je pokrýt konfigurační a stavová data a řídící operace specifické pro Knot Resolver. Jedná se také o funkce Knot Resolveru a odlišné způsoby konfigurace dané především jeho modulární architekturou. Je tedy logické, že každá jednotlivá implementace DNS resolveru by mohla mít podobný rozšiřující YANG modul, který pokrývá specifické vlastnosti dané implementace DNS resolveru.

Modul se specifickou konfigurací Knot Resolveru je přímo závislý na společném datovém modelu pro DNS resolvery `cznic-resolver-common` (kapitola 8.2), který tento modul rozšiřuje a tudíž není jej možné použít samostatně. Rozšíření je prováděno pomocí augmentací již existujícího uzlu ve společném modelu a následným přidáním uzlů specifických pro Knot Resolver v dané kategorii nebo podstromu. Celkový vzhled datového modelu ve stromovém schématu (YANG Tree Diagram [12]) je možné vidět v příloze A. Uzly definované tímto rozšiřujícím YANG modulem mají před názvem uzlu jméno modulu, případně jsou součástí modulu definujícího uzel o úroveň výše. V následujících kapitolách jsou detailně vysvětleny důležité části tohoto rozšiřujícího modulu.

8.3.1 Konfigurace a stav procesů

Knot Resolver je realizován několika samostatnými procesy (kapitola 2), `kresd` a `kres-cache-gc`, ke kterým v návrhu (kapitola 6) přibývá nový proces `kres-watcher` (kapitola 6.3). Pro tento proces je v datovém modelu určena především kategorie `server`, kde je počáteční konfigurace pro tyto procesy při startu operačního systému. Jedná se o názvy jednotlivých instancí kresd, kolik instancí má být spuštěno a zda má být spuštěn automaticky i cache garbage collector (kres-cache-gc).

```
module: cznic-resolver-common
   +--rw dns-resolver
      +--rw server
         +--ro package-version?                       string
         +--rw user-name?                             string
         +--rw group-name?                            string
         +--rw cznic-resolver-knot:auto-start?        boolean <false>
         +--rw cznic-resolver-knot:auto-cache-gc?     boolean <true>
         +--rw cznic-resolver-knot:kresd-instances?   uint8 <1>
         +--ro cznic-resolver-knot:status
            +--ro cznic-resolver-knot:kres-instances* [name]
            |  +--ro cznic-resolver-knot:name         string
            |  +--ro cznic-resolver-knot:status?      process-status
            +--ro cznic-resolver-knot:cache-gc?       process-status
```

Zdrojový kód 8.2: YANG Tree Diagram - startovní konfigurace procesů Knot Resolveru

- **auto-start**

  Značí, zda mají být spuštěny procesy Knot Resolveru automaticky při startu serveru/systému. Jaké procesy to jsou určují následující parametry níže.

- **auto-cache-gc**

  Značí, zda se má `kres-cache-gc` proces spouštět/zastavovat automaticky s instancemi `kresd`.

- **kresd-instances**

  Počet instancí `kresd` procesu, které mají být spuštěny při startu Knot Resolveru. Natavení na hodnotu `0` značí, že bude spouštět stejný počet instancí jako je dostupných jader/vláken CPU serveru.

- **status**

  Stavová data určená k informaci o stavu jednotlivých instancí *kresd* procesu a `kres-cache-gc` procesu zda jsou aktivní či nikoli.

8.3.2 Instance kresd procesu

Architektura Knot Resolveru umožňuje spuštění několika instancí `kresd` procesu zároveň (kapitola 2.2). Tyto instance mohou operovat s odlišnou konfigurací, proto je datový model rozšířen o seznam s konfigurací pro jednotlivé instance. Podstrom datového modelu pro konfiguraci jednotlivých instancí zrcadlí globální konfiguraci včetně rozšíření. Výjimku tvoří uzel `server`, který `kresd` proces nepoužívá a uzel `cache`, který je konfigurován pouze na globální úrovni. Jednotlivé instance v základu používají globální konfiguraci sdílenou mezi všemi instancemi. Pokud je v seznamu přítomna speciální konfigurace pro danou instanci, má tato konfigurace pro tuto instanci vyšší prioritu než konfigurace globální.

```
module: cznic-resolver-common
   +--rw dns-resolver
      +--rw cznic-resolver-knot:instances* [name]
         +--rw cznic-resolver-knot:name      string
         |   ...
         +---x cznic-resolver-knot:start
         +---x cznic-resolver-knot:stop
         +---x cznic-resolver-knot:restart
```

Zdrojový kód 8.3: YANG Tree Diagram - seznam kresd instancí se specifickou konfigurací

Instance `kresd` používají ke své identifikaci své jméno `name`, které jednotlivé procesy získají pomocí spuštění přes `systemd` službu. Jako další rozšíření jsou k jednotlivým instancím přidány akce (actions), což jsou operace prováděné nad uzly datového modelu. Zde umožňují řídit jednotlivé konkrétní instance `kresd` procesu samostatně.

8.3.3 kres-cache-gc

Konfigurace údržbáře cache paměti `kres-cache-gc` je umístěna v datovém modelu v kategorii `cache`, kde má svůj vlastní podstrom začínající uzlem `garbage-collector` a obsahující jeho kompletní konfiguraci. Navíc jsou v modelu přidány akce (actions) pro samostatné řízení `kres-cache-gc`.

```
module: cznic-resolver-common
+--rw dns-resolver
    +--rw cache
        +--rw cznic-resolver-knot:garbage-collector
            +--ro cznic-resolver-knot:version?          string
            |   ...
            +---x cznic-resolver-knot:start
            +---x cznic-resolver-knot:stop
            +---x cznic-resolver-knot:restart
```

Zdrojový kód 8.4: YANG Tree Diagram - datový model pro `kres-cache-gc` proces

8.3.4 Logování a monitorování

Jazyk YANG umožňuje definovat strukturu dat pro asynchronní zprávy zvané jako notifikace. Tento způsob zpráv je využitý pro zaznamenávání logů do souboru v `sysrepo` a nebo pro monitorování DNS resolveru díky real-time sledování příchozích zpráv pomocí knihovny `sysrepo` v `kresctl` (kapitola 6.4) nebo NETCONF protokolu. V případě navrhovaného systému pro Knot Resolver může jít například o informování, že byly spuštěny požadované procesy nebo o poslání chybové hlášky.

```
module: cznic-resolver-common
   +--rw dns-resolver
      +--rw logging
         +---n cznic-resolver-knot:log
            +--ro cznic-resolver-knot:log-type?     string
            +--ro cznic-resolver-knot:time-stamp?   timestamp
            +--ro cznic-resolver-knot:process?      string
            +--ro cznic-resolver-knot:message?      string
```

Zdrojový kód 8.5: YANG Tree Diagram - YANG notifikace určená k logování

8.3.5 Komentáře

Komentáře k navrženému YAML konfiguračnímu souboru (kapitola 6.4.2) jsou v datovém modelu definovány pomocí rozšiřujícího YANG modulu `ietf-yang-metadata` (RFC 7952 [37]) jako anotace k jednotlivým uzlům. Díky tomu může být komentář v konfiguračním souboru převeden na typ uzlu `annotation` a vložen společně s konfigurací do datového úložiště `sysrepo` jako metadata. Výsledkem je, že tyto komentáře jsou k dispozici i po exportu konfigurace z datového úložiště.

```
ietf-yang-metadata:annotation comment {
    type string;
    description
        "Annotation definition for comments.";
}
```

Zdrojový kód 8.6: YANG - definice komentáře pomocí YANG anotace (RFC 7952 [37]

8.4 Moduly kresd procesu

V kapitole 5.1.1 bylo zmíněno, že jednotlivé moduly určující funkcionality `kresd` se musí před použitím nejdříve manuálně načíst. Jazyk YANG umožňuje označit uzel vlastností `presence`, kdy jeho samotná přítomnost v konfiguračních datech může značit zapnutí určité funkcionality. V YANG stromovém schématu (RFC 8340 [12]) je takový uzel označen znakem `'!'`. Jednotlivé moduly pro `kresd` by se měli automaticky načítat při přítomnosti daného uzlu v konfiguračních datech. Jako příklad je uvedena kategorie pro DNS64, která tak automaticky zapíná funkcionalitu a `kresd` proces by měl modul automaticky načíst před provedením jeho konfigurace.

```
module: cznic-resolver-common
   +--rw dns-resolver
     +--rw dns64!
        +--rw prefix?   ietf-inet-types:ipv6-prefix <64:ff9b::/96>
```

Zdrojový kód 8.7: YANG Tree Diagram - DNS64 funkcionalita

9 Testování

Elementární testování bylo prováděno k ověření základní funkcionality pokaždé, když byla implementována nová funkce do systému. K ověření využití dynamicky alokované paměti byl používán AddressSanitizer nástroje `Clang`[11] při kompilaci Knot Resolveru pomocí `meson` build systému. Pokud bylo potřeba debugovat byl k ladění použit obslužný program `gdb`[12]. K testování `sysrepo` funkcionalit implementovaných v procesech byly používány nástroje `sysrepoctl` a `sysrepocfg`. Při návrhu a tvorbě datového modelu byl pro validaci YANG modulů a konfiguračních dat využíván nástroj `yanglint`, který je součástí knihovny `libyang` [40].

9.1 sysrepo import a export

Jedná se o měření času potřebného k importu konfigurační dat do datového úložiště sysrepo a exportu těchto dat z operačního úložiště. Měření bylo provedeno na žádost vývojového týmu Knot Resolverů, jelikož v budoucnu bude potřeba dynamicky konfigurovat velké i velké datové sety. Pro potřeby měření byl vytvořen testovací datový model, který je realizovaný YANG modulem `cznic-test-sysrepo`. Tento datový model představuje záznamy RPZ (Response Policy Zones [56], kapitola 1.10), které mohou v souborech dosahovat počtů až několika milionů. Jedná se o extrémní případ, ale bylo potřeba zjistit jak `sysrepo` v této situaci obstojí.

```
module: cznic-test-sysrepo
   +--rw tests
      +--rw items* [name]
         +--rw name       string
         +--rw actions*   string
         +--rw ids*       uint64
```

Zdrojový kód 9.1: YANG Tree Diagram - datový model pro testování

[11] `https://clang.llvm.org/docs/AddressSanitizer.html`

[12] `https://www.gnu.org/software/gdb/`

Pro generování testovacích dat byl vytvořen skript `data_generator.py` v jazyce python. Skript generuje libovolný počet položek seznamu `items` datového modelu do souboru ve formátu JSON. Například `python3 data_generator.py -n 50000` vygeneruje soubor s 50 tisíci položkami v seznamu.

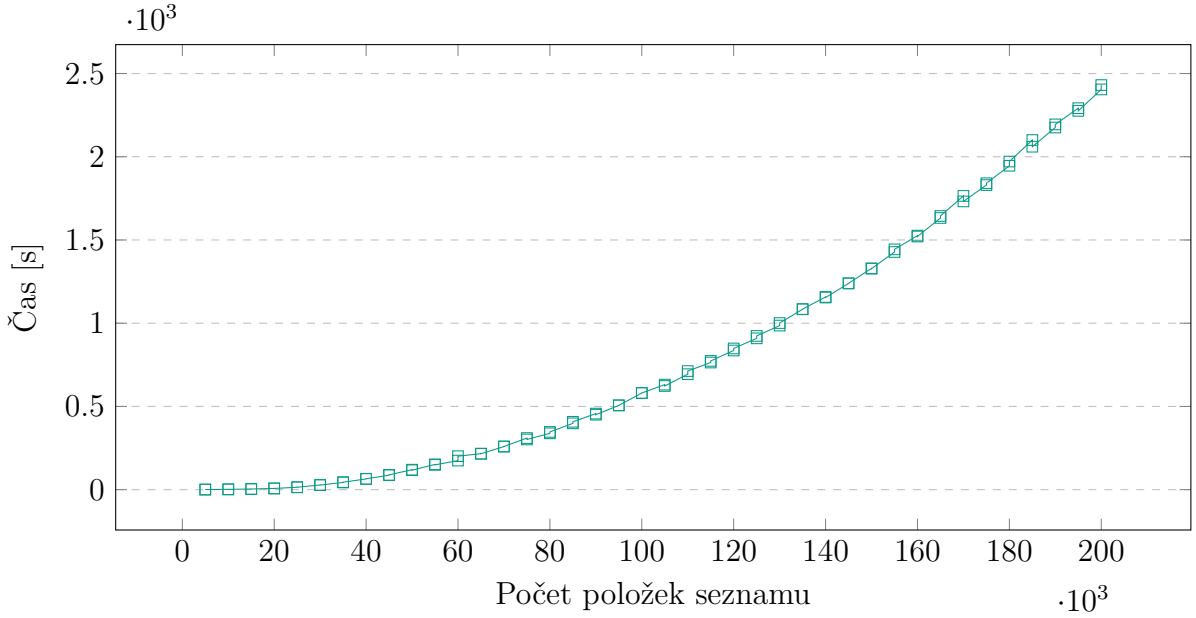
Testovací skript `sysrepo_bench_stats.sh` (viz přílohy) provádí měření doby importu a následného exportu dat ze `sysrepo` pro sekvenci velikostí seznamu konfiguračních dat. Měření začíná na 5000 položkách seznamu a zvyšuje se sekvenčně o 5000 až do 200000 položek. Měření pro velikosti konfiguračních dat je prováděno dvakrát.

9.1.1 Výsledky měření

Operační systém:	Arch Linux
	5.4.36-1-MANJARO
Procesor:	Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz
Paměť:	15GiB System memory

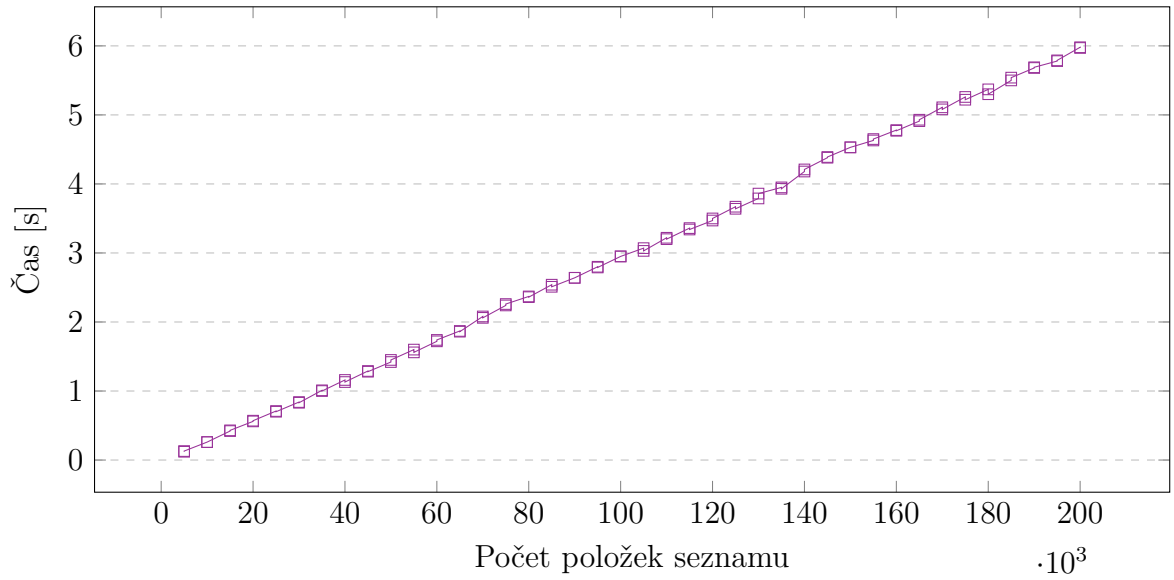
Tabulka 9.1: Specifikace testovacího prostředí

Import dat Jedná se o kvadratickou časovou složitost $\mathcal{O}(n^2)$, která je způsobena validací konfiguračních dat. V seznamu (YANG list) je totiž potřeba kontrolovat unikátnost jednotlivých záznamů, každá nová položka seznamu musí být při validaci porovnána se všemi původními položkami v seznamu.



Graf 9.1: Doba importu konfiguračních dat do sysrepo v závislosti na počtu položek.

Export dat Jedná se o lineární časovou složitost $\mathcal{O}(n)$, jelikož operační data při exportu nejsou validována.



Graf 9.2: Doba exportu operačních dat ze sysrepo v závislosti na počtu položek.

Vyhodnocení Z měření je jasné, že v současnosti není výkonnost `sysrepo`, především při importu extrémně velkých datových setů, na optimální úrovni. Jedná se opravdu o extrémní případ, který ale může především v blacklist seznamech jako je RPZ nastat (kapitola 1.10). Vývojáři nástrojů `sysrepo` a `libyang` jsou si tohoto problému vědomi a již mají naplánovanou optimalizaci v dalších verzích těchto knihoven.

9.2 Případ využití netopeer2

Po vytvoření prototypu bylo potřeba vyzkoušet, zda je možné Knot Resolver konfigurovat vzdáleně pomocí NETCONF protokolu. K tomu je potřeba nainstalovat nástroj `netopeer2` na server společně s Knot Resolverem (kapilota 7.6) a NETCONF klienta na zařízení ze kterého bude Knot Resolver konfigurovaný.

9.2.1 Instalace netopeer2 serveru

Netopeer2 server využívá knihovny `sysrepo`, `libyang` a `libnetconf2`. Instalace `sysrepo` a `libyang` je popsána v kapitole 7.1. Pro ověření, že kompilace proběhla správně se osvědčilo před systémovou instalací spustit testy `make test`, k tomu jsou vyžadovány nástroje pro testování `cmocka` a `valgrind`.

```
$ git clone https://github.com/CESNET/libnetconf2.git
$ cd libnetconf2 && git checkout devel
$ mkdir build && cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
$ make
$ make test
# make install
```

Zdrojový kód 9.2: Instalace knihovny libnetconf2 [16]

```
$ git clone https://github.com/CESNET/netopeer2.git
$ cd netopeer2 && git checkout devel-server
$ mkdir server/build && cd server/build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
$ make
$ make test
# make install
```

Zdrojový kód 9.3: Instalace NETCONF serveru netopeer2 [16]

`netopeer2-server` je bezprostředně po úspěšné instalaci možné spustit nejlépe v debug módu pomocí parametru *-d*.

```
$ netopeer2-server -d
```

Konfigurace `netopeer2` server má svou vlastní konfiguraci uloženou přímo v `sysrepo`. To znamená, že je možné ji upravovat přímo v sysrepo nebo vzdáleně prostřednictvím NETCONF protokolu. Ve výchozím nastavení je server nastaven pouze aby poslouchal zprávy přicházející přes SSH protokol na portu 830.

9.2.2 Klient

Nejlepší je nejprve klienta nainstalovat na serveru, kde je připravený `netopeer2-server` pro ověření funkčnosti na `localhost` a až poté se pokusit o vzdálené připojení.

```
$ git clone https://github.com/CESNET/netopeer2.git
$ cd netopeer2 && git checkout devel-server
$ mkdir cli/build && cd cli/build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
$ make
# make install
```

Zdrojový kód 9.4: Instalace NETCONF klienta netopeer2-cli [16]

Připojení Veškeré dostupné možnosti připojení na server je možné vidět v operačních datech YANG modulu `ietf-netconf-server` zadáním následujícího příkazu. Příkaz je potřeba zadat na serveru.

```
$ sysrepocfg -X -m ietf-netconf-server -d operational -f json
```

Nyní je možné se pomocí `netopeer2-cli` klienta připojit na server. Jako výchozí protokol je používáno `ssh`, `port` je 830 a `hostname` je `localhost`. Pomocí příkazu `status` je možné kdykoli vidět stav aktuálního připojení k serveru.

```
$ netopeer2-cli
> connect -h
> connect --host localhost --port 830 --login <username>
> status
```

Pokud je připojení úspěšné, je možné zadat příkaz `get` k získání dat ze serveru.

```
> get --filter-xpath /cznic-resolver-common/*
```

10 Shrnutí

Tato kapitola je shrnutím výsledného systému a jeho porovnání z hlediska řízení, konfigurace a monitoringu s původním stavem Knot Resolveru a jeho procesů a rozebírá jaké věci se v návrhu systému podařilo zlepšit a které výhody původního systému naopak chybí. Na konci kapitoly je představen možný postup při navázání na výsledky této práce.

Výhody nového systému

- Bylo vytvořeno společné administrační rozhraní `kresctl` (kapitola 6.4) pro všechny části a procesy Knot Resolveru. Řízení, konfigurace a monitoring lze provádět z jediného rozhraní.
- Byl využit jasně definovaný mechanismus administrace (NETCONF protokol), který zásadně zjednodušuje administraci. Knot Resolver je nyní lokálně a vzdáleně konfigurovatelný pomocí jakéhokoli klienta podporujícího NETCONF protokol.
- V původním systému nebylo možné `kres-cache-gc` 2.3.1) vůbec konfigurovat za běhu a jednotlivé instance `kresd` musely být konfigurované samostatně (kapitola 2.2). Nyní je možné v novém systému procesy konfigurovat za běhu a všechny `kresd` instance zároveň.
- Lokální i vzdálená konfigurace probíhá prostřednictvím jednoho rozhraní datového úložiště `sysrepo` (kapitola 6).
- Konfiguraci je možné validovat oproti datovému modelu v jazyce YANG (kapitola 8) aniž by musela být spuštěna.
- Byl vytvořen čitelnější, přehlednější a srozumitelnější deklarativní formát konfigurace pro administrátory v jazyce YAML (kapitola 6.4.2).
- Automatické načítání modulů pro `kresd` proces (kapitola 8.4) zjednodušuje administraci.

Nevýhody nového systému

- Nový systém zatím nepokrývá kompletní konfiguraci Knot Resolveru a jeho procesů. Datový model se dá ale jednoduše rozšířit a potřebnou funkci do systému přidat.
- Pomalý import velkých datových setů do datového úložiště `sysrepo` (kapitola 9.1). Problém se nachází v knihovnách `libyang` a `sysrepo`, tím pádem jejich pozdější optimalizace povedou automaticky ke zlepšení výkonu bez nutnosti zásahu do kódu systému.

10.1 Budoucnost projektu

Výstupem této diplomové práce je počáteční prototyp systému, na který bude další vývoj a případné reálné nasazení teprve navazovat. Proto je důležité zmínit, jak a jakým směrem může další vývoj tohoto systému pro řízení, konfiguraci a monitoring Knot Resolveru a jeho procesů směřovat.

Datový model vytvořený v rámci práce nepokrývá kompletní konfiguraci Knot Resolveru, ale pouze jeho vybranou část (kapitola 8). Proto bude potřeba datový model postupně rozšiřovat a implementovat do procesů pokud má být systém reálně nasazen. Administrační rozhraní `kresctl` je na datovém modelu částečně nezávislé, jde tedz především o implementaci rozšíření do jednotlivých procesů v podobě zpětných volání (callbacks) pro odběr nových uzlů.

Některé části výsledného systému by bylo možné navrhnout univerzálně, především konfigurační rozhraní `kresctl` a data v YAML formátu. Jednalo by se tedy o jakéhosi univerzálního konfiguračního klienta pro `sysrepo` a možnost importu a export dat v YAML formátu.

Po úspěšném nasazení systému pro Knot Resolver budou zkoumány i možnosti integrace s jinými implementacemi DNS resolveru. S tímto záměrem byl také navrhován společný datový model pro DNS resolvery (kapitola 8.2). Výsledkem by byla částečně kompatibilní konfigurace a nebo umožnění administrace všech těchto implementací pomocí NETCONF protokolu zároveň z jedné centrální stanice.

Závěr

Tato diplomová práce se zabývá návrhem systému pro lokální i vzdálenou administraci open-source DNS resolveru Knot Resolver, který se vyznačuje značnou modularitou a je realizován několika samostatnými procesy konfigurovanými jazykem Lua. Hlavním cílem bylo navrhnout a realizovat koncept systému, který zlepšuje v současnosti problematické řízení, konfiguraci a monitorování Knot Resolveru a jeho procesů. Dílčím cílem bylo do navrhovaného systému zakomponovat i případné obecné zlepšení vzdálené administrace různých implementací DNS resolveru, které je způsobené nekompatibilitou jejich administračních rozhraní a rozdílnou konfigurací.

Úvodní část práce představuje základy protokolů a technologií, které byly potřebné k návrhu a samotné realizaci systému nastudovat. Rešerše DNS protokolu a architektury Knot Resolveru byly důležité k pochopení problematické administrace Knot Resolveru. NETCONF protokol společně s jazykem YANG a sadou nástrojů libyang, sysrepo, netopeer2 byly zase klíčové při analýze problémů a schopnosti navrhnout jejich řešení. Po nastudování tématu byly s vedoucím a konzultantem práce probrány přímo konkrétní problémy administrace Knot Resolveru. Byla také probrána problematika vztahující se obecně ke vzdálené administraci a konfiguraci DNS resolverů, která byla nakonec zakomponována přímo do řešení systému. Na základě těchto informací byl vytvořen seznam požadavků a návrhy na řešení k jednotlivým problémům.

Požadavky na nový systém a navržená řešení byla velmi důležitá při návrhu samotného systému. Ukázalo se, že většina problémů nebo požadavků je velmi dobře řešitelná s pomocí datového modelu a datového úložiště pro konfiguraci sysrepo. Datový model reprezentuje konfiguraci jednotlivých procesů Knot Resolveru uložených v sysrepo a parametry jednotlivých operací prováděné přes NETCONF protokol. Při návrhu datového modelu se podařilo jeho základ definovat pomocí společného datového modelu pro DNS resolver, který je založený na průniku konfigurace tří open-source implementací včetně Knot Resolveru. Tento sjednocený základ byl následně rozšířen o konfiguraci a operace specifické pro Knot Resolver. Pro řízení jednotlivých procesů byl vytvořen nový proces nazvaný jako `kres-watcher`. který reaguje na povely přes RPC operace definované datovým modelem.

Výsledným přínosem této práce je návrh základního prototypu systému, který zjednodušuje řízení, konfiguraci a monitorování Knot Resolveru a jeho procesů prostřednictvím vytvořeného rozhraní `kresctl`. Díky tomu, že je celý systém založen na standardizovaném a otevřeném protokolu NETCONF, je možné Knot Resolver konfigurovat také vzdáleně bez složité konfigurace na straně serveru a klienta. Přínosem do budoucna by mohl být také použitý datový model v jazyce YANG, který je založen na průniku konfigurací open-source DNS resolverů. To by umožnilo částečnou kompatibilitu konfigurace mezi nimi a tím velmi zjednodušilo jejich konfiguraci. Navržený systém realizuje veškeré dílčí požadavky a cíle, zadání práce lze tedy považovat za splněné.

Seznam použité literatury

1. CHISHOLM, S.; TREVINO, H. *NETCONF Event Notifications* [Internet Requests for Comments]. RFC Editor, 2008. ISSN 2070-1721. RFC. RFC Editor.
2. *Against The Use Of Programming Languages in Configuration Files* [online] [cit. 2020-04-22]. Dostupné z: `https://taint.org/2011/02/18/001527a.html`.
3. AITCHISON, Ron. *Pro DNS and BIND 10*. Second Edition. Apress, 2011. ISBN 1430230487.
4. ARENDS, R.; AUSTEIN, R.; LARSON, M.; MASSEY, D.; ROSE, S. *DNS Security Introduction and Requirements* [Internet Requests for Comments]. RFC Editor, 2005. ISSN 2070-1721. Dostupné také z: `http://www.rfc-editor.org/rfc/rfc4033.txt`. RFC. RFC Editor. `http://www.rfc-editor.org/rfc/rfc4033.txt`.
5. ARENDS, R.; AUSTEIN, R.; LARSON, M.; MASSEY, D.; ROSE, S. *Protocol Modifications for the DNS Security Extensions* [Internet Requests for Comments]. RFC Editor, 2005. ISSN 2070-1721. Dostupné také z: `http://www.rfc-editor.org/rfc/rfc4035.txt`. RFC. RFC Editor. `http://www.rfc-editor.org/rfc/rfc4035.txt`.
6. ARENDS, R.; AUSTEIN, R.; LARSON, M.; MASSEY, D.; ROSE, S. *Resource Records for the DNS Security Extensions* [Internet Requests for Comments]. RFC Editor, 2005. ISSN 2070-1721. Dostupné také z: `http://www.rfc-editor.org/rfc/rfc4034.txt`. RFC. RFC Editor. `http://www.rfc-editor.org/rfc/rfc4034.txt`.
7. BAGNULO, M.; SULLIVAN, A.; MATTHEWS, P.; BEIJNUM, I. van. *DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers* [Internet Requests for Comments]. RFC Editor, 2011. ISSN 2070-1721. RFC. RFC Editor.
8. BAO, C.; HUITEMA, C.; BAGNULO, M.; BOUCADAIR, M.; LI, X. *IPv6 Addressing of IPv4/IPv6 Translators* [Internet Requests for Comments]. RFC Editor, 2010. ISSN 2070-1721. RFC. RFC Editor.

9. BIERMAN, A.; BJORKLUND, M.; WATSEN, K. *RESTCONF Protocol* [Internet Requests for Comments]. RFC Editor, 2017. ISSN 2070-1721. RFC. RFC Editor.

10. BJORKLUND, M. *The YANG 1.1 Data Modeling Language* [Internet Requests for Comments]. RFC Editor, 2016. ISSN 2070-1721. RFC. RFC Editor.

11. BJORKLUND, M. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)* [Internet Requests for Comments]. RFC Editor, 2010. ISSN 2070-1721. Dostupné také z: `http://www.rfc-editor.org/rfc/rfc6020.txt`. RFC. RFC Editor. `http://www.rfc-editor.org/rfc/rfc6020.txt`.

12. BJORKLUND, M.; BERGER, L. *YANG Tree Diagrams* [Internet Requests for Comments]. RFC Editor, 2018. ISSN 2070-1721. BCP. RFC Editor.

13. BJORKLUND, M.; SCHOENWAELDER, J.; SHAFER, P.; WATSEN, K.; WILTON, R. *NETCONF Extensions to Support the Network Management Datastore Architecture* [Internet Requests for Comments]. RFC Editor, 2019. ISSN 2070-1721. RFC. RFC Editor.

14. BJORKLUND, M.; SCHOENWAELDER, J.; SHAFER, P.; WATSEN, K.; WILTON, R. *Network Management Datastore Architecture (NMDA)* [Internet Requests for Comments]. RFC Editor, 2018. ISSN 2070-1721. RFC. RFC Editor.

15. *CESNET/libyang* [online]. CESNET, 2020 [cit. 2020-03-04]. Dostupné z: `https://github.com/CESNET/libyang`. original-date: 2015-03-13T10:25:52Z.

16. *CESNET/Netopeer2* [online]. CESNET, 2020 [cit. 2020-03-04]. Dostupné z: `https://github.com/CESNET/Netopeer2`. original-date: 2015-12-03T10:05:33Z.

17. CIMPANU, Catalin. *First-ever malware strain spotted abusing new DoH (DNS over HTTPS) protocol* [online]. 2019 [cit. 2020-03-01]. Dostupné z: `https://www.zdnet.com/article/first-ever-malware-strain-spotted-abusing-new-doh-dns-over-https-protocol/`.

18. *CZ.NIC: Správce domény CZ*. Dostupné také z: `https://nic.cz/`.

19. *CZ.NIC - Otevřené DNSSEC Validující Resolvery* [online] [cit. 2020-04-22]. Dostupné z: `https://www.nic.cz/odvr/`.

20. *CZ-NIC/knot-resolver* [online]. CZ.NIC, 2020 [cit. 2020-02-29]. Dostupné z: `https://github.com/CZ-NIC/knot-resolver`. original-date: 2015-03-31T12:01:38Z.

21. *DNS over TLS vs. DNS over HTTPS* [online] [cit. 2020-05-11]. Dostupné z: `https://www.cloudflare.com/learning/dns/dns-over-tls/`.

22. ENNS, R. *NETCONF Configuration Protocol* [Internet Requests for Comments]. RFC Editor, 2006. ISSN 2070-1721. Dostupné také z: `http://www.rfc-editor.org/rfc/rfc4741.txt`. RFC. RFC Editor. `http://www.rfc-editor.org/rfc/rfc4741.txt`.

23. ENNS, R.; BJORKLUND, M.; SCHOENWAELDER, J.; BIERMAN, A. *Network Configuration Protocol (NETCONF)* [Internet Requests for Comments]. RFC Editor, 2011. ISSN 2070-1721. Dostupné také z: `http://www.rfc-editor.org/rfc/rfc6241.txt`. RFC. RFC Editor. `http://www.rfc-editor.org/rfc/rfc6241.txt`.

24. FUJIWARA, K.; KATO, A.; KUMARI, W. *Aggressive Use of DNSSEC-Validated Cache* [Internet Requests for Comments]. RFC Editor, 2017. ISSN 2070-1721. RFC. RFC Editor.

25. HARDAKER, W. *Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs)* [Internet Requests for Comments]. RFC Editor, 2006. ISSN 2070-1721. RFC. RFC Editor.

26. HARRENSTIEN, K.; STAHL, M.K.; FEINLER, E.J. *Hostname Server* [Internet Requests for Comments]. RFC Editor, 1985. ISSN 2070-1721. RFC. RFC Editor.

27. HARRENSTIEN, K.; WHITE, V.; FEINLER, E.J. *Hostnames Server* [Internet Requests for Comments]. RFC Editor, 1982. ISSN 2070-1721. RFC. RFC Editor.

28. HOFFMAN, P.; MCMANUS, P. *DNS Queries over HTTPS (DoH)* [Internet Requests for Comments]. RFC Editor, 2018. ISSN 2070-1721. RFC. RFC Editor.

29. HU, Z.; ZHU, L.; HEIDEMANN, J.; MANKIN, A.; WESSELS, D.; HOFFMAN, P. *Specification for DNS over Transport Layer Security (TLS)* [Internet Requests for Comments]. RFC Editor, 2016. ISSN 2070-1721. RFC. RFC Editor.

30. HUBERT, Bert. *Centralised DoH is bad for privacy, in 2019 and beyond* [online]. 2019 [cit. 2020-03-01]. Dostupné z: `https://blog.powerdns.com/2019/09/25/centralised-doh-is-bad-for-privacy-in-2019-and-beyond/`.

31. *IANA — Root Servers* [online] [cit. 2020-03-01]. Dostupné z: https://www.iana.org/domains/root/servers.

32. *IANA — Root Zone Database* [online] [cit. 2020-05-12]. Dostupné z: https://www.iana.org/domains/root/db.

33. *IntroductionToDBus* [online]. 2013 [cit. 2020-02-18]. Dostupné z: https://www.freedesktop.org/wiki/IntroductionToDBus/.

34. KERRISK, Michael. *The SO_REUSEPORT socket option [LWN.net]* [online]. 2013 [cit. 2020-02-29]. Dostupné z: https://lwn.net/Articles/542629/.

35. *Knot Resolver Documentation*. 2020. Dostupné také z: https://knot-resolver.readthedocs.io.

36. LAURIE, B.; SISSON, G.; ARENDS, R.; BLACKA, D. *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence* [Internet Requests for Comments]. RFC Editor, 2008. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc5155.txt. RFC. RFC Editor. http://www.rfc-editor.org/rfc/rfc5155.txt.

37. LHOTKA, L. *Defining and Using Metadata with YANG* [Internet Requests for Comments]. RFC Editor, 2016. ISSN 2070-1721. RFC. RFC Editor.

38. LHOTKA, L. *JSON Encoding of Data Modeled with YANG* [Internet Requests for Comments]. RFC Editor, 2016. ISSN 2070-1721. RFC. RFC Editor.

39. LHOTKA, Ladislav. Knot Resolver slaví páté narozeniny. *Https://blog.nic.cz/*. 2019. Dostupné také z: https://blog.nic.cz/2019/06/27/knot-resolver-slavi-pate-narozeniny/.

40. *libyang: About* [online] [cit. 2020-03-04]. Dostupné z: https://netopeer.liberouter.org/doc/libyang/master/.

41. LIU, Cricket; ALBITZ, Paul. *DNS and BIND*. 5th ed. Sebastopol, CA: O'Reilly, 2006. ISBN 0596100574.

42. MAKAREM, Christopher. *How DNSSEC Works* [online]. 2018 [cit. 2020-02-20]. Dostupné z: https://medium.com/iocscan/how-dnssec-works-9c652257be0.

43. MOCKAPETRIS, P. *Domain names: Concepts and facilities* [Internet Requests for Comments]. RFC Editor, 1983. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc882.txt. RFC. RFC Editor. http://www.rfc-editor.org/rfc/rfc882.txt.

44. MOCKAPETRIS, P. *Domain names - concepts and facilities* [Internet Requests for Comments]. RFC Editor, 1987. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc1034.txt. STD. RFC Editor. http://www.rfc-editor.org/rfc/rfc1034.txt.

45. MOCKAPETRIS, P. *Domain names - implementation and specification* [Internet Requests for Comments]. RFC Editor, 1987. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc1035.txt. STD. RFC Editor. http://www.rfc-editor.org/rfc/rfc1035.txt.

46. *Netmod Status Pages* [online] [cit. 2020-03-03]. Dostupné z: https://tools.ietf.org/wg/netmod/.

47. *PowerDNS Recursor Settings — PowerDNS Recursor documentation* [online] [cit. 2020-03-11]. Dostupné z: https://doc.powerdns.com/recursor/settings.html.

48. SCHOENWAELDER, J. *Overview of the 2002 IAB Network Management Workshop* [Internet Requests for Comments]. RFC Editor, 2003. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc3535.txt. RFC. RFC Editor. http://www.rfc-editor.org/rfc/rfc3535.txt.

49. SALOWEY, J.; ZHOU, H.; ERONEN, P.; TSCHOFENIG, H. *Transport Layer Security (TLS) Session Resumption without Server-Side State* [Internet Requests for Comments]. RFC Editor, 2008. ISSN 2070-1721. Dostupné také z: http://www.rfc-editor.org/rfc/rfc5077.txt. RFC. RFC Editor. http://www.rfc-editor.org/rfc/rfc5077.txt.

50. STJOHNS, M. *Automated Updates of DNS Security (DNSSEC) Trust Anchors* [Internet Requests for Comments]. RFC Editor, 2007. ISSN 2070-1721. STD. RFC Editor.

51. *sysrepo/sysrepo* [online] [cit. 2020-05-21]. Dostupné z: https://github.com/sysrepo/sysrepo.

52. *systemd* [online]. 2020 [cit. 2020-03-04]. Dostupné z: https://systemd.io/.

53. *systemd.service* [online] [cit. 2020-03-04]. Dostupné z: https://www.freedesktop.org/software/systemd/man/systemd.service.html.

54. *The Meson Build system* [online] [cit. 2020-04-27]. Dostupné z: https://mesonbuild.com/.

55. *Unbound - unbound.conf.5* [online] [cit. 2020-03-11]. Dostupné z: https://nlnetlabs.nl/documentation/unbound/unbound.conf/.

56. VIXIE, Paul; SCHRYVER, Vernon. *DNS Response Policy Zones (RPZ)* [Working Draft]. 2017. Dostupné také z: http://www.ietf.org/internet-drafts/draft-ietf-dnsop-dns-rpz-00.txt. Internet-Draft. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-ietf-dnsop-dns-rpz-00.txt.

57. WEILER, S.; IHREN, J. *Minimally Covering NSEC Records and DNSSEC On-line Signing* [Internet Requests for Comments]. RFC Editor, 2006. ISSN 2070-1721. RFC. RFC Editor.

Seznam obrázků, tabulek a zdrojových kódů

Seznam obrázků

Seznam tabulek

Seznam grafů

Seznam zdrojových kódů

Seznam zkratek

ARPA The Advanced Research Projects Agency.

ARPANET ARPA Network.

CLI Command Line Interface.

DARPA The Defense Advanced Research Projects Agency.

DNS Domain Name System.

DS Delegation Signer.

IANA Internet Assigned Numbers Authority.

IETF Internet Engineering Task Force.

IPC InterProcess Communication.

ISC Internet System Consortium.

ISP Internet Service Provider.

KSK Key Signing Key.

NSEC3 Next Secure Record.

RPC Remote Procedure Call.

RR Resource Record.

RRSIG Resource Record Signature.

SNMP Simple Network Management Protocol.

SSH Secure Shell.

TCP Transmission Control Protocol.

TLD Top-Level Domain.

TLS Transport Layer Security.

TTL Time to Live.

UDP User Datagram Protocol.

XML Extensible Markup Language.

YANG Yet Another Next Generation.

ZSK Zone Signing Key.

Přílohy

Některé přílohy související s touto diplomovou prací jsou k dispozici pouze v elektronické podobě. Adresářová struktura příloh v elektronické podobě je následující.

- `knot-resolver/` ..... zdrojový kód Knot Resolveru
  - `modules/sysrepo/common/` ..... společný zdrojový kód pro procesy
  - `utils/` ..... zdrojové kódy procesů
- `yang-model/` ..... YANG datový model
  - `example-data/` ..... příklady konfiguračních dat
  - `yang-modules/` ..... YANG moduly
  - `common-config.ods` ..... společná konfigurace implementací DNS resolveru
- `sysrepo-test/` ..... zdrojové kódy určené k testování sysrepo
  - `yang-modules/` ..... testovací YANG moduly
  - `sysrepo_bench_stats.sh` ..... sysrepo testovací skript

A YANG schéma datového modelu

```
module: cznic-resolver-common
+--rw dns-resolver
    +--rw server
    |  +--ro package-version?                        string
    |  +--rw user-name?                              string
    |  +--rw group-name?                             string {set-group}?
    |  +--rw cznic-resolver-knot:auto-start?         boolean <false>
    |  +--rw cznic-resolver-knot:auto-cache-gc?      boolean <true>
    |  +--rw cznic-resolver-knot:kresd-instances?    uint8 <1>
    |  +--ro cznic-resolver-knot:status
    |     +--ro cznic-resolver-knot:kres-instances* [name]
    |     |  +--ro cznic-resolver-knot:name          string
    |     |  +--ro cznic-resolver-knot:status?       process-status
    |     +--ro cznic-resolver-knot:cache-gc?        process-status
    +--rw network
    |  +--rw listen-interfaces* [id]
    |  |  +--rw id                           string
    |  |  +--rw ip-address                   ietf-inet-types:ip-address
    |  |  +--rw port?                        ietf-inet-types:port-number <53>
    |  |  +--rw cznic-resolver-knot:kind?    dns-transport-protocol <dns>
    |  +--rw source-address
    |  |  +--rw ipv4?   ietf-inet-types:ipv4-address-no-zone
    |  |  +--rw ipv6?   ietf-inet-types:ipv6-address-no-zone
    |  +--rw client-transport
    |  |  +--rw l2-protocols?   l2-protocol-selection
    |  +--rw recursion-transport
    |  |  +--rw l2-protocols?   l2-protocol-selection
    |  +--rw udp-payload-size?      uint16 <4096>
    |  +--rw tls
    |     +--rw cert?                                fs-path
    |     +--rw cert-key?                            fs-path
    |     +--rw cznic-resolver-knot:sticket-secret?  string
    +--rw resolver
    |  +--rw stub-zones* [domain]
    |  |  +--rw domain        ietf-inet-types:domain-name
    |  |  +--rw nameserver?   ietf-inet-types:host
    |  |  +--rw port?         ietf-inet-types:port-number <53>
    |  +--rw hints
    |  |  +--rw root-hint* [name]
    |  |  |  +--rw name      ietf-inet-types:domain-name
    |  |  |  +--rw values*   ietf-inet-types:ip-address-no-zone
    |  |  +--rw root-zone-file?                   fs-path
    |  |  +--rw cznic-resolver-knot:hint* [name]
    |  |  |  +--rw cznic-resolver-knot:name
    |     ↪  ietf-inet-types:domain-name
    |  |  |  +--rw cznic-resolver-knot:values*
    |     ↪  ietf-inet-types:ip-address-no-zone
    |  |  |  +--rw cznic-resolver-knot:canonical?   boolean <false>
```

```
47      |  |  +--rw cznic-resolver-knot:hosts-file?
        ↪  cznic-resolver-common:fs-path
48      |  +--rw options
49      |     +--rw glue-checking?        boolean <true>
50      |     +--rw qname-minimisation?   boolean
51      |     +--rw query-loopback?       boolean <true>
52      |     +--rw reorder-rrset?        boolean <false>
53      +--rw dnssec!
54      |  +--rw trust-anchors* [domain]
55      |  |  +--rw domain             ietf-inet-types:domain-name
56      |  |  +--ro key-file?          fs-path
57      |  |  +--rw auto-update?       boolean <true>
58      |  |  +--rw trust-anchor* [id]
59      |  |  |  +--rw id                       uint8
60      |  |  |  +--rw (trust-anchor-rdata)?
61      |  |  |     +--:(ds)
62      |  |  |     |  +--rw ds
63      |  |  |     |     +--rw key-tag        uint16
64      |  |  |     |     +--rw algorithm
        ↪  cznic-dns-parameters:dnssec-algorithm
65      |  |  |     |     +--rw digest-type
        ↪  cznic-dns-parameters:digest-algorithm
66      |  |  |     |     +--rw digest         cznic-dns-parameters:hex-digits
67      |  |  |     +--:(dnskey)
68      |  |  |        +--rw dnskey
69      |  |  |           +--rw flags?        cznic-dns-parameters:dnskey-flags
70      |  |  |           +--rw protocol?     uint8 <3>
71      |  |  |           +--rw algorithm
        ↪  cznic-dns-parameters:dnssec-algorithm
72      |  |  |           +--rw public-key    binary
73      |  |  +---x add-trust-anchor
74      |  |     +--rw input
75      |  |        +---w (trust-anchor-rdata)?
76      |  |           +--:(ds)
77      |  |           |  +---w ds
78      |  |           |     +---w key-tag        uint16
79      |  |           |     +---w algorithm
        ↪  cznic-dns-parameters:dnssec-algorithm
80      |  |           |     +---w digest-type
        ↪  cznic-dns-parameters:digest-algorithm
81      |  |           |     +---w digest         cznic-dns-parameters:hex-digits
82      |  |           +--:(dnskey)
83      |  |              +---w dnskey
84      |  |                 +---w flags?       cznic-dns-parameters:dnskey-flags
85      |  |                 +---w protocol?    uint8 <3>
86      |  |                 +---w algorithm
        ↪  cznic-dns-parameters:dnssec-algorithm
87      |  |                 +---w public-key   binary
88      |  +--rw negative-trust-anchors*   ietf-inet-types:domain-name
89      +--rw dns64!
90      |  +--rw prefix?   ietf-inet-types:ipv6-prefix <64:ff9b::/96>
```

```
+--rw logging
|  +--rw verbosity?                uint8 <1>
|  +---n cznic-resolver-knot:log
|     +--ro cznic-resolver-knot:log-type?     string
|     +--ro cznic-resolver-knot:time-stamp?   ietf-yang-types:timestamp
|     +--ro cznic-resolver-knot:process?      string
|     +--ro cznic-resolver-knot:message?      string
+--rw cache
|  +--ro current-size?                           uint64
|  +--rw max-size?                               uint64
|  +--rw min-ttl?                                uint32 <0>
|  +--rw max-ttl?                                uint32 <172800>
|  +--rw cznic-resolver-knot:storage?
|  ↪ cznic-resolver-common:fs-path </var/cache/knot-resolver>
|  +--rw cznic-resolver-knot:prefill* [origin]
|  |  +--rw cznic-resolver-knot:origin
|  ↪ ietf-inet-types:domain-name
|  |  +--rw cznic-resolver-knot:url                ietf-inet-types:uri
|  |  +--rw cznic-resolver-knot:ca-file
|  ↪ cznic-resolver-common:fs-path
|  |  +--rw cznic-resolver-knot:refresh-interval?   uint32 <86400>
|  +--rw cznic-resolver-knot:garbage-collector
|     +--ro cznic-resolver-knot:version?               string
|     +--rw cznic-resolver-knot:interval?              uint32 <1000>
|     +--rw cznic-resolver-knot:threshold?             percent <80>
|     +--rw cznic-resolver-knot:release-percentage?    percent <10>
|     +--rw cznic-resolver-knot:temporary-keys-space?  uint64
|     +--rw cznic-resolver-knot:rw-items?              uint64 <100>
|     +--rw cznic-resolver-knot:rw-duration?           uint64
|     +--rw cznic-resolver-knot:rw-delay?              uint64
|     +--rw cznic-resolver-knot:dry-run?               boolean <false>
|     +---x cznic-resolver-knot:start
|     +---x cznic-resolver-knot:stop
|     +---x cznic-resolver-knot:restart
+--rw cznic-resolver-knot:instances* [name]
   +--rw cznic-resolver-knot:name        string
   +--ro cznic-resolver-knot:status?      process-status
   +--rw cznic-resolver-knot:network
   |  +--rw cznic-resolver-knot:listen-interfaces* [id]
   |  |  +--rw cznic-resolver-knot:id          string
   |  |  +--rw cznic-resolver-knot:ip-address  ietf-inet-types:ip-address
   |  |  +--rw cznic-resolver-knot:port?  ietf-inet-types:port-number <53>
   |  |  +--rw cznic-resolver-knot:kind?  dns-transport-protocol <dns>
   |  +--rw cznic-resolver-knot:source-address
   |  |  +--rw cznic-resolver-knot:ipv4?
   |  ↪ ietf-inet-types:ipv4-address-no-zone
   |  |  +--rw cznic-resolver-knot:ipv6?
   |  ↪ ietf-inet-types:ipv6-address-no-zone
   |  +--rw cznic-resolver-knot:client-transport
   |  |  +--rw cznic-resolver-knot:l2-protocols?
   |  ↪ cznic-resolver-common:l2-protocol-selection
```

```
136 |  +--rw cznic-resolver-knot:recursion-transport
137 |  |  +--rw cznic-resolver-knot:l2-protocols?
    ↪  cznic-resolver-common:l2-protocol-selection
138 |  +--rw cznic-resolver-knot:udp-payload-size?      uint16 <4096>
139 |  +--rw cznic-resolver-knot:tls
140 |     +--rw cznic-resolver-knot:cert?
    ↪  cznic-resolver-common:fs-path
141 |     +--rw cznic-resolver-knot:cert-key?
    ↪  cznic-resolver-common:fs-path
142 |     +--rw cznic-resolver-knot:sticket-secret?   string
143 +--rw cznic-resolver-knot:resolver
144 |  +--rw cznic-resolver-knot:stub-zones* [domain]
145 |  |  +--rw cznic-resolver-knot:domain
    ↪  ietf-inet-types:domain-name
146 |  |  +--rw cznic-resolver-knot:nameserver?   ietf-inet-types:host
147 |  |  +--rw cznic-resolver-knot:port?
    ↪  ietf-inet-types:port-number <53>
148 |  +--rw cznic-resolver-knot:hints
149 |  |  +--rw cznic-resolver-knot:root-hint* [name]
150 |  |  |  +--rw cznic-resolver-knot:name
    ↪  ietf-inet-types:domain-name
151 |  |  |  +--rw cznic-resolver-knot:values*
    ↪  ietf-inet-types:ip-address-no-zone
152 |  |  +--rw cznic-resolver-knot:root-zone-file?
    ↪  cznic-resolver-common:fs-path
153 |  |  +--rw cznic-resolver-knot:hint* [name]
154 |  |  |  +--rw cznic-resolver-knot:name
    ↪  ietf-inet-types:domain-name
155 |  |  |  +--rw cznic-resolver-knot:values*
    ↪  ietf-inet-types:ip-address-no-zone
156 |  |  |  +--rw cznic-resolver-knot:canonical?   boolean <false>
157 |  |  +--rw cznic-resolver-knot:hosts-file?
    ↪  cznic-resolver-common:fs-path
158 |  +--rw cznic-resolver-knot:options
159 |     +--rw cznic-resolver-knot:glue-checking?        boolean <true>
160 |     +--rw cznic-resolver-knot:qname-minimisation?   boolean
161 |     +--rw cznic-resolver-knot:query-loopback?       boolean <true>
162 |     +--rw cznic-resolver-knot:reorder-rrset?        boolean <false>
163 +--rw cznic-resolver-knot:dnssec!
164 |  +--rw cznic-resolver-knot:trust-anchors* [domain]
165 |  |  +--rw cznic-resolver-knot:domain
    ↪  ietf-inet-types:domain-name
166 |  |  +--ro cznic-resolver-knot:key-file?
    ↪  cznic-resolver-common:fs-path
167 |  |  +--rw cznic-resolver-knot:auto-update?       boolean <true>
168 |  |  +--rw cznic-resolver-knot:trust-anchor* [id]
169 |  |  |  +--rw cznic-resolver-knot:id                        uint8
170 |  |  |  +--rw (cznic-resolver-knot:trust-anchor-rdata)?
171 |  |  |     +--:(cznic-resolver-knot:ds)
172 |  |  |     |  +--rw cznic-resolver-knot:ds
173 |  |  |     |     +--rw cznic-resolver-knot:key-tag        uint16
```

```
174       |  |  |     |     +--rw cznic-resolver-knot:algorithm
          ↪  cznic-dns-parameters:dnssec-algorithm
175       |  |  |     |     +--rw cznic-resolver-knot:digest-type
          ↪  cznic-dns-parameters:digest-algorithm
176       |  |  |     |     +--rw cznic-resolver-knot:digest
          ↪  cznic-dns-parameters:hex-digits
177       |  |  |     +--:(cznic-resolver-knot:dnskey)
178       |  |  |        +--rw cznic-resolver-knot:dnskey
179       |  |  |           +--rw cznic-resolver-knot:flags?
          ↪  cznic-dns-parameters:dnskey-flags
180       |  |  |           +--rw cznic-resolver-knot:protocol?     uint8 <3>
181       |  |  |           +--rw cznic-resolver-knot:algorithm
          ↪  cznic-dns-parameters:dnssec-algorithm
182       |  |  |           +--rw cznic-resolver-knot:public-key    binary
183       |  |  +---x cznic-resolver-knot:add-trust-anchor
184       |  |     +--rw cznic-resolver-knot:input
185       |  |        +---w (cznic-resolver-knot:trust-anchor-rdata)?
186       |  |           +--:(cznic-resolver-knot:ds)
187       |  |           |  +---w cznic-resolver-knot:ds
188       |  |           |     +---w cznic-resolver-knot:key-tag        uint16
189       |  |           |     +---w cznic-resolver-knot:algorithm
          ↪  cznic-dns-parameters:dnssec-algorithm
190       |  |           |     +---w cznic-resolver-knot:digest-type
          ↪  cznic-dns-parameters:digest-algorithm
191       |  |           |     +---w cznic-resolver-knot:digest
          ↪  cznic-dns-parameters:hex-digits
192       |  |           +--:(cznic-resolver-knot:dnskey)
193       |  |              +---w cznic-resolver-knot:dnskey
194       |  |                 +---w cznic-resolver-knot:flags?
          ↪  cznic-dns-parameters:dnskey-flags
195       |  |                 +---w cznic-resolver-knot:protocol?     uint8 <3>
196       |  |                 +---w cznic-resolver-knot:algorithm
          ↪  cznic-dns-parameters:dnssec-algorithm
197       |  |                 +---w cznic-resolver-knot:public-key    binary
198       |  +--rw cznic-resolver-knot:negative-trust-anchors*
          ↪  ietf-inet-types:domain-name
199       +--rw cznic-resolver-knot:dns64!
200       |  +--rw cznic-resolver-knot:prefix?   ietf-inet-types:ipv6-prefix
          ↪  <64:ff9b::/96>
201       +--rw cznic-resolver-knot:logging
202       |  +--rw cznic-resolver-knot:verbosity?   uint8 <1>
203       +---x cznic-resolver-knot:start
204       +---x cznic-resolver-knot:stop
205       +---x cznic-resolver-knot:restart
206
207  rpcs:
208    +---x start
209    +---x stop
210    +---x restart
```

B Lua - Knot Resolver konfigurace

```
local systemd_instance = os.getenv("SYSTEMD_INSTANCE")

if string.match(systemd_instance, '^dns') then
    net.listen('127.0.0.1', 53, { kind = 'dns' })
    net.listen('185.43.135.1', 53, { kind = 'dns' })
    net.listen('193.17.47.1', 53, { kind = 'dns' })
    net.listen('217.31.204.134', 53, { kind = 'dns' })
    net.listen('::1', 53, { kind = 'dns' })
    net.listen('2001:148f:ffff::1', 53, { kind = 'dns' })
    net.listen('2001:148f:fffe::1', 53, { kind = 'dns' })
    net.listen('2001:1488:800:400::2:134', 53, { kind = 'dns' })
elseif string.match(systemd_instance, '^tls') then
    net.listen('127.0.0.1', 853, { kind = 'tls' })
    net.listen('185.43.135.1', 853, { kind = 'tls' })
    net.listen('193.17.47.1', 853, { kind = 'tls' })
    net.listen('217.31.204.134', 853, { kind = 'tls' })
    net.listen('::1', 853, { kind = 'tls' })
    net.listen('2001:148f:ffff::1', 853, { kind = 'tls' })
    net.listen('2001:148f:fffe::1', 853, { kind = 'tls' })
    net.listen('2001:1488:800:400::2:134', 853, { kind = 'tls' })
elseif string.match(systemd_instance, '^doh') then
    net.listen('127.0.0.1', 443, { kind = 'doh' })
    net.listen('185.43.135.1', 443, { kind = 'doh' })
    net.listen('193.17.47.1', 443, { kind = 'doh' })
    net.listen('217.31.204.134', 443, { kind = 'doh' })
    net.listen('::1', 443, { kind = 'doh' })
    net.listen('2001:148f:ffff::1', 443, { kind = 'doh' })
    net.listen('2001:148f:fffe::1', 443, { kind = 'doh' })
    net.listen('2001:1488:800:400::2:134', 443, { kind = 'doh' })
else
    panic("Use kresd@dns*, kresd@tls@ or kresd@doh* instance names")
end

-- Load useful modules
modules = {
    'hints > iterate',     -- Load /etc/hosts and allow custom root hints
    'stats',               -- Track internal statistics
    'predict',             -- Prefetch expiring/frequent records
    'rebinding < iterate', -- Protect from rebinding attack
    'nsid',                -- Get instance ID to DNS client, RFC 5001
}

-- Cache size
cache.size = 1000 * MB

-- NSID --
nsid.name('134@'..systemd_instance)

-- DoT
```

```
50 if string.match(systemd_instance, '^tls') then
51     net.tls("/etc/ssl/nic-certs/odvr.nic.cz/odvr.nic.cz.chained.crt",
52             "/etc/ssl/nic-certs/odvr.nic.cz/odvr.nic.cz.key")
53 end
54
55
56 -- DoH
57 if string.match(systemd_instance, '^doh') then
58     modules.load('http')
59     http.config({
60         tls = true,
61         cert = "/etc/ssl/nic-certs/odvr.nic.cz/odvr.nic.cz.chained.crt",
62         key = "/etc/ssl/nic-certs/odvr.nic.cz/odvr.nic.cz.key"
63     }, 'doh')
64 end
65
66 -- Watchdog
67 watchdog.config({ qname = 'udp53.cz.', qtype = kres.type.A })
68
69 stats_prev = stats.list()
70 function get_stat_increment()
71     local stats_now = stats.list()
72     local doh_increment = (
73         stats_now['request.doh'] - stats_prev['request.doh'])
74     local dot_increment = (stats_now['request.dot'] -
    ↪  stats_prev['request.dot'])
75     local internal_increment = (stats_now['request.internal'] -
    ↪  stats_prev['request.internal'])
76     local answer_c_increment = ( stats_now['answer.cached'] -
    ↪  stats_prev['answer.cached'])
77     local total_increment = (stats_now['request.total'] -
    ↪  stats_prev['request.total'] - internal_increment)
78     stats_prev = stats_now
79     return {doh=doh_increment,dot=dot_increment,
    ↪  total=total_increment,answer_c=answer_c_increment}
80 end
81
82 trust_anchors.set_insecure({'csas.cz', 'cezonline.cez.cz',
 ↪  'pacient-soap.erecept.sukl.cz', 'lekar-soap.erecept.sukl.cz',
 ↪  'new-anycast.odvr.cz' })
```

C JSON - Knot Resolver konfigurace

```
{
    "cznic-resolver-common:dns-resolver": {
        "server": {
            "cznic-resolver-knot:auto-start": true,
            "cznic-resolver-knot:auto-cache-gc": true,
            "cznic-resolver-knot:kresd-instances": 2
        },
        "network": {
            "listen-interfaces": [
                {
                    "id": 0,
                    "ip-address": "127.0.0.1"
                },
                {
                    "id": 1,
                    "ip-address": "::1"
                }
            ],
            "source-address": {
                "ipv6": "2001:db8:0:2::1"
            },
            "client-transport": {
                "l2-protocols": "ipv6"
            },
            "recursion-transport": {
                "l2-protocols": "ipv4 ipv6"
            },
            "udp-payload-size": 4096
        },
        "resolver": {
            "stub-zones": [
                {
                    "domain": "stub.example.com",
                    "nameserver": "192.0.2.1",
                    "port": 53
                },
                {
                    "domain": "stub.example.net",
                    "nameserver": "198.51.100.1"
                }
            ],
            "hints": {
                "cznic-resolver-knot:hint": [
                    {
                        "name": "localhost",
                        "canonical": true,
                        "values": [
                            "127.0.0.1",
                            "::1"
```

```
                        ]
                    }
                ],
                "cznic-resolver-knot:hosts-file": "/etc/hosts",
                "root-hint": [
                    {
                        "name": "a.root-servers.net",
                        "values": [
                            "198.41.0.4",
                            "2001:503:ba3e::2:30"
                        ]
                    }
                ],
                "root-zone-file": "/etc/resolver/root.hints"
            },
            "options": {
                "glue-checking": true,
                "qname-minimisation": true,
                "reorder-rrset": true,
                "query-loopback": true
            }
        },
        "dnssec": {
            "trust-anchors": [
                {
                    "domain": ".",
                    "auto-update": true,
                    "trust-anchor": [
                        {
                            "id": 0,
                            "ds": {
                                "algorithm": "RSASHA256",
                                "digest": "49AAC11D7B6F6446702E54A160737160
                                ↪  7A1A41855200FD2CE1CDDE32F24E8FB5",
                                "digest-type": "SHA-256",
                                "key-tag": 19036
                            }
                        }
                    ]
                }
            ],
            "negative-trust-anchors": [
                "bad.example.com",
                "worse.example.com"
            ]
        },
        "dns64": {
            "prefix": "64:ff9b::/96"
        },
        "logging": {
            "verbosity": 2
```

```
        },
        "cache": {
            "cznic-resolver-knot:storage": "/var/cache/knot-resolver",
            "max-size": 104857600,
            "min-ttl": 50,
            "max-ttl": 172800,
            "cznic-resolver-knot:prefill": [
                {
                  "origin": ".",
                  "url": "https://www.internic.net/domain/root.zone",
                  "ca-file": "/etc/pki/tls/certs/ca-bundle.crt"
                }
            ],
            "cznic-resolver-knot:garbage-collector": {
                "interval": 1000,
                "threshold": 90,
                "release-percentage": 20,
                "temporary-keys-space": 0,
                "rw-items": 0,
                "rw-duration": 0,
                "rw-delay": 0,
                "dry-run": false
            }
        },
        "cznic-resolver-knot:instances": [
            {
                "name": "dot",
                "network": {
                    "listen-interfaces": [
                        {
                            "id": 1,
                            "ip-address": "198.51.100.1",
                            "port": 553,
                            "cznic-resolver-knot:kind": "dot"
                        }
                    ],
                    "tls": {
                        "cert": "server.crt",
                        "cert-key": "server.key",
                        "sticket-secret": "b4ZfPnEa"
                    }
                },
                "logging": {
                    "verbosity": 5
                }
            }
        ]
    }
}
```

D YAML - Knot Resolver konfigurace

```yaml
---
server:
    auto-start: true
    auto-cache-gc: true
    kresd-instances: 2
network:
    listen-interfaces:
      - id: 0
        ip-address: 127.0.0.1
      - id: 1
        ip-address: "::1"
    source-address:
        ipv6: 2001:db8:0:2::1
    client-transport:
        l2-protocols: ipv6
    recursion-transport:
        l2-protocols: ipv4 ipv6
    udp-payload-size: 4096
resolver:
    stub-zones:
      - domain: stub.example.com
        nameserver: 192.0.2.1
        port: 53
      - domain: stub.example.net
        nameserver: 198.51.100.1
    hints:
        hint:
          - name: localhost
            canonical: true
            values:
              - 127.0.0.1
              - "::1"
          - name: loopback
            values:
              - 127.0.0.1
        hosts-file: "/etc/hosts"
        root-hint:
          - name: a.root-servers.net
            values:
              - 198.41.0.4
              - 2001:503:ba3e::2:30
        root-zone-file: "/etc/resolver/root.hints"
    options:
        glue-checking: true
        qname-minimisation: true
        reorder-rrset: true
        query-loopback: true
dnssec:
    trust-anchors:
```

```
      - domain: "."
        auto-update: true
        trust-anchor:
          - id: 0
            ds:
            algorithm: RSASHA256
            digest: 49AAC11D7B6F6446702E54A1607371607A1A41855200FD2CE1CDDE3
             ↪  2F24E8FB5
            digest-type: SHA-256
            key-tag: 19036
    negative-trust-anchors:
        - bad.example.com
        - worse.example.com
dns64:
    prefix: 64:ff9b::/96
logging:
    verbosity: 2
cache:
    storage: "/var/cache/knot-resolver"
    max-size: 104857600
    min-ttl: 50
    max-ttl: 172800
    prefill:
      - origin: "."
        url: https://www.internic.net/domain/root.zone
        ca-file: "/etc/pki/tls/certs/ca-bundle.crt"
    garbage-collector:
        interval: 1000
        threshold: 90
        release-percentage: 20
        temporary-keys-space: 0
        rw-items: 0
        rw-duration: 0
        rw-delay: 0
        dry-run: false
instances:
  - name: dot
    network:
        listen-interfaces:
          - id: 1
            ip-address: 198.51.100.1
            port: 553
            kind: dot
        tls:
            cert: server.crt
            cert-key: server.key
        sticket-secret: b4ZfPnEa
    logging:
        verbosity: 5
```