VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

KONFIGURAČNÍ API KNIHOVNY LIBNETCONF2 PODLE YANG MODELU IETF-NETCONF-SERVER

CONFIGURATION API OF THE LIBNETCONF2 LIBRARY ACCORDING TO THE IETF-NETCONF-SERVER YANG MODEL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ROMAN JANOTA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JIŘÍ MATOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

154233

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Janota Roman**
Program:	Informační technologie
Název:	**Konfigurační API knihovny libnetconf2 podle YANG modelu ietf-netconf-server**
Kategorie:	Počítačové sítě
Akademický rok:	2023/24

Zadání:

1. Seznamte se s konfiguračním API knihovny libnetconf2 a porovnejte jej s parametry v YANG modelu ietf-netconf-server.
2. Navrhněte nové konfigurační API umožňující nastavení všech stávajících parametrů serveru způsobem kompatibilním s YANG modelem ietf-netconf-server.
3. Implementujte a zdokumentujte navržené API.
4. Vytvořte a aplikujte sadu testů ověřujících funkčnost výsledného řešení.
5. Zrealizujte případovou studii použití implementovaného API v rámci netopeer2 NETCONF serveru.
6. Zhodnoťte zkušenosti získané při realizaci případové studie a diskutujte možnosti dalšího vylepšení API.

Literatura:

- Dle pokynů vedoucího a konzultanta práce.

Při obhajobě semestrální části projektu je požadováno:
Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Matoušek Jiří, Ing., Ph.D.**
Konzultant:	Mgr. Michal Vaško
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	30.10.2023

Abstrakt

Cílem této práce je navrhnout a implementovat nové uživatelské rozhraní pro knihovnu *libnetconf2*, které bude umožňovat nakonfigurovat NETCONF server podle popisu *ietf-netconf-server* YANG modelu. Podstatou řešení byla prvně analýza knihovny a modelu a jejich následné porovnání. Na základě výsledků porovnání jsem navrhl nové konfigurační rozhraní, jehož vstupem jsou YANG data popisující konfiguraci serveru. Navržené řešení umožňuje nastavovat chování serveru dvěma způsoby. První způsob zachovává a upravuje stávající konfiguraci pomocí speciálního atributu operace, zatímco druhý způsob kompletně nahrazuje stávající konfiguraci novou. Nové řešení je dále rozděleno do dvou fází — vytvoření a aplikace konfiguračních dat. Práce se dále zaměřuje na implementaci návrhu, na jeho nedostatky, na které jsem narazil až při implementaci, a následně na testování, které bylo provedeno dvěma způsoby, a to pomocí vlastní testovací sady a následně integrací nového rozhraní do existujícího *open-source* NETCONF serveru s názvem *netopeer2*. V práci dále popisuji svůj přínos k *open-source* projektu *libssh* a k samotnému návrhu YANG modelu *ietf-netconf-server*. Výsledky této práce umožňují uživatelům knihovny *libnetconf2* nakonfigurovat svůj NETCONF server podle standardizovaného popisu nebo sdílet svou konfiguraci pomocí konfiguračních dat. Nové konfigurační rozhraní je nyní součástí hlavní větve projektu *libnetconf2*.

Abstract

The aim of this thesis is to propose and implement a new application programming interface for the *libnetconf2* library, which allows for configuration of a NETCONF server based on the *ietf-netconf-server* YANG model. The approach begins with an analysis of both the library and the model, followed by their comparison. Based on the results of the comparison, I then designed a new configuration interface, which takes YANG data describing the NETCONF server configuration as input. The proposed solution enables configuring the server in two ways. The former approach preserves the existing configuration and adjusts it based on a special operation attribute. The latter approach entirely replaces the previous configuration with the new one. The proposed solution comprises of two phases — the creation and the application of configuration data. The focus then shifts to implementation, identifying flaws in the design that arose during implementation, and testing, which was initially done using my own test suite and then using an existing open-source NETCONF server called *netopeer2*. Additionally, this thesis describes my contribution to an open-source project *libssh* as well as to the *ietf-netconf-server* YANG model draft itself. The primary outcome of this work is the the ability for users of the *libnetconf2* library to configure their NETCONF server in a standardized manner, as well as the ability to share the NETCONF server configuration in the form of configuration data. The new configuration interface is now part of the *libnetconf2*'s main branch.

Klíčová slova

konfigurace, API, libnetconf2, NETCONF, YANG

Keywords

configuration, API, libnetconf2, NETCONF, YANG

Citace

JANOTA, Roman. *Konfigurační API knihovny libnetconf2 podle YANG modelu ietf-netconf-server*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Matoušek, Ph.D.

Konfigurační API knihovny libnetconf2 podle YANG modelu ietf-netconf-server

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Matouška, Ph.D. Další informace mi poskytl Mgr. Michal Vaško ze sdružení CESNET, který byl odborným konzultantem této práce. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Roman Janota
6. května 2024

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Jiřímu Matouškovi, Ph.D. za jeho cenné rady a doporučení při vedení mé bakalářské práce. Taktéž bych rád poděkoval svému konzultantovi Mgr. Michalovi Vaškovi za seznámení s knihovnou *libnetconf2*, za trpělivost, ochotu a pomoc.

Obsah

Kapitola 1

Úvod

Knihovna *libnetconf2* je projektem, který svým uživatelům umožňuje abstrakci detailů při implementaci svého NETCONF (*Network Configuration Protocol*) klienta nebo serveru. NETCONF je síťový protokol definující jednoduchý mechanismus, kterým mohou být spravována síťová zařízení, a to pomocí zasílání standardizovaných zpráv. V této práci se zaměřím na návrh a implementaci uživatelského rozhraní knihovny, které je kompatibilní se standardizovaným popisem možností konfigurace NETCONF serveru. Tento standardizovaný popis je součástí YANG modelu s názvem *ietf-netconf-server*, jehož obsah je v práci porovnán se stávajícím konfiguračním rozhraním knihovny. Vývoj knihovny se z velké části odvíjí od zpětné vazby uživatelů, součástí níž byla i žádost o možnost konfigurace serveru podle standardizovaného popisu. V současnosti je knihovna *libnetconf2* spravována skupinou zabývající se nástroji pro monitoring a konfiguraci síťových zařízení sdružení CESNET.

Jedna z výhod kompatibility se standardizovaným popisem je následující. Ačkoliv bylo původní konfigurační rozhraní knihovny kvalitně promyšlené a z uživatelského hlediska přívětivé, tak se vyskytl i případ užití, pro který bylo toto rozhraní nevyhovující. Jedná se o případ, kdy rozhraní knihovny není využíváno člověkem, ale programem. Tento program rozumí popisu modelu a dokáže pomocí něj vytvořit konfigurační data. Tato data by nejraději předal knihovně, aby je zpracovala a nakonfigurovala podle nich server, ale místo toho musí tato data projít sám a postupně podle nich používat jednotlivé funkce z rozhraní knihovny. Tímto programem je další projekt sdružení CESNET s názvem *netopeer2*.

Součástí této práce je návrh nového konfiguračního rozhraní. Tento návrh je založen na aplikaci konfiguračních dat, jež jsou vstupem nového rozhraní. Ovšem tato data musí vyhovovat popisu modelu *ietf-netconf-server*[1]. V práci se zaměřím i na problematiku tvorby těchto dat pro běžného uživatele. Data jsou aplikována pomocí jedné funkce rozhraní a výsledkem je nový stav *libnetconf2* serveru.

Výstupy této práce už jsou nyní volně dostupné uživatelům a implementaci spolu s její dokumentací je možné najít v online repozitáři[2] projektu *libnetconf2*. Nicméně při implementaci jsem narazil na to, že některé části návrhu nebyly dokonalé, a tak se v této práci věnuji i problémům, které vyvstaly z chyb v návrhu. V blízké budoucnosti bude hlavním cílem uživatelská podpora, která je majoritně tvořena opravováním nahlášených chyb a pomáháním s přechodem na novou verzi knihovny.

Kapitola 2, která následuje hned po úvodu, nejprve do jisté míry seznámí čtenáře se základy modelovacího jazyka YANG. Následně bude čtenář seznámen se síťovým protokolem

[1] Dále v textu je čtenář seznámen s tím, že tato konfigurační data mohou být popsána i dvěma jinými modely.

[2] Dostupné z `https://github.com/CESNET/libnetconf2`.

NETCONF a s komunikací v rámci tohoto protokolu. V kapitole 3 bude představeno původní rozhraní knihovny *libnetconf2* spolu s YANG modelem *ietf-netconf-server*. Následně tato kapitola obsahuje porovnání tohoto rozhraní s popisem modelu. Kapitola 4 se zabývá návrhem nového rozhraní, který vychází z poznatků získaných z porovnání. Následuje kapitola 5, jejímž účelem je seznámit čtenáře s implementací nového rozhraní. V této kapitole se mimo jiné také věnuji problémům s návrhem, vlastnímu rozšíření modelu *ietf-netconf-server* a dalším změnám v knihovně, které se netýkaly konfigurace. Kapitola 6 se zabývá testováním implementace pomocí vlastní testovací sady. Předposlední kapitola 7 popisuje případovou studii použití nového rozhraní v rámci *netopeer2* NETCONF serveru. V poslední kapitole 8 jsou shrnuty poznatky, nastíněna možná rozšíření do budoucna a popsány zkušenosti, kterých jsem nabyl při dělání na této práci. Příloha A je věnována programu *yanglint*, s nímž jsem pracoval jak při zmíněném porovnání, tak při implementaci nového konfiguračního rozhraní. A na závěr příloha B popisuje obsah přiloženého paměťového média.

Kapitola 2

Modelovací jazyk YANG a protokol NETCONF

V celé mé práci se budou často vyskytovat dvě slova — NETCONF a YANG. Účelem této kapitoly je vysvětlit oba pojmy a popsat to, co jsem z nich ve své práci využil. Při vývoji protokolu NETCONF bylo otázkou, jak standardně definovat data, která budou reprezentovat stav a komunikaci tohoto protokolu. Odpovědí na tuto otázku byl modelovací jazyk YANG.

2.1 Modelovací jazyk YANG

Jazyk YANG byl původně navržený primárně pro modelování konfiguračních a stavových dat protokolu NETCONF. Jeho nejnovější standardizovanou verzí je 1.1, která je popsána v RFC (*Request For Comments*)[1] 7950 [2]. Byl navržen a standardizován organizací IETF (*Internet Engineering Task Force*), což je organizace vydávající hlavně standardy pro síťové aplikace a komunikaci. Jazyk YANG má poměrně jednoduchou syntax, a proto je i dobře čitelný.

Model popsaný jazykem YANG je hierarchický a má stromovou strukturu. Každý uzel v tomto stromu má svůj název a obsahuje buď hodnotu, nebo množinu uzlů, jež jsou jeho potomky. YANG podporuje modularitu, tedy umožňuje importovat definice z jiných modelů. Jazyk YANG dále dovoluje modelu pozměnit jiný importovaný model. Tento princip například umožňuje uživatelům rozšířit a přizpůsobit si standardizované modely.

V jazyku YANG uvažujeme dva druhy pohledu na daný model — schéma a data. Schéma je kostra, která udává, jak mají vypadat data. Schéma tedy například popisuje datový typ uzlu. Hodnota daného uzlu ve validních YANG datech pak musí být tohoto typu. Jazyk YANG rozděluje dva druhy typů — vestavěné a derivované. Příkladem vestavěného typu může být *int8* (osmibitové celé číslo) nebo třeba *string* (řetězec). Derivované typy jsou typy odvozené z vestavěných typů nebo z jiných derivovaných typů, a to většinou za pomoci nějakých restrikcí na hodnoty, kterých může daný typ nabývat. Příkladem derivovaného typu může například být typ „IP adresa“.

Důležitou součástí jazyka YANG jsou tzv. *features*, které se dají chápat jako sada nějakých volitelných vlastností nebo rozšíření modelu. Tato rozšíření umožňují podmínečně rozšiřovat či případně měnit stromovou strukturu stejného nebo jiného modelu o nějaké další uzly. Při pohledu na YANG model je tedy možné zvolit si množinu vlastností, které

[1] Série dokumentů popisující internetové standardy, protokoly a jiné. Dále už pouze jako RFC.

mají být podporované, a od této zvolené množiny se následně odvíjí i výsledná struktura daného YANG modelu.

Běžný model popsaný v jazyku YANG by se dal rozdělit na hlavičku a tělo. Každý model musí mít své jméno, které je potřeba definovat ještě před hlavičkou. Hlavička nese základní informace o daném modelu. Mezi povinné položky hlavičky patří tzv. *namespace* (tj. jmenný prostor) modelu, což je globálně unikátní URI (*Uniform Resource Identifier*), tedy identifikátor jednoznačně identifikující daný model, který se používá při serializaci YANG dat do formátu XML (*Extensible Markup Language*) [19]. Další povinnou položkou je prefix, který se dá chápat jako zkratka jmenného prostoru modelu a který se používá například při importování daného modelu v jiném modelu nebo opět při serializaci do XML. Hlavička dále může obsahovat datum revize, popis toho, co daný model popisuje, použitou verzi jazyku YANG a mimo jiné i informace o autorovi daného modelu. Dále mohou následovat konstrukce pro importování a exportování definic anebo definice derivovaných typů.

V těle modelu se nachází už samotný hierarchický popis, který může obsahovat několik různých konstrukcí, ale zaměřím se zde pouze na tři z nich. První konstrukcí je kontejner (anglicky *container*). Kontejner má svůj název a rozlišujeme dva typy — tzv. *presence* a *non-presence*. *Non-presence* kontejner nemá sám o sobě žádný konfigurační význam a existuje pouze kvůli zapouzdření jeho potomků. Tedy jeho existence nijak neovlivňuje chování konfigurovaného systému. V této práci je vždy slovem „kontejner“ myšlen tento typ, pokud není explicitně řečeno jinak. Naopak *presence* kontejner má nějaký význam a dá se představit jako jeden bit reprezentující dvě hodnoty — buď existuje, anebo neexistuje. I *presence* kontejner může zapouzdřovat své potomky.

Druhou konstrukcí je list (anglicky *leaf*). Uzel typu list je ve schéma listovým uzlem, který má svůj název, typ a nemá žádné další potomky. V YANG datech se pak může nebo nemusí vyskytovat. U uzlu typu list rozlišujeme dva důležité atributy, které může mít. List je možné definovat s výchozí hodnotou, což znamená, že pokud se v YANG datech nenachází, nabývá výchozí hodnoty. Dále uzel typu list může být povinný, což znamená, že se v YANG datech vyskytovat musí.

Poslední konstrukcí je seznam (anglicky *list*). Uzel typu seznam zapouzdřuje své potomky a může v YANG datech existovat v několika instancích, kde každá instance je jednoznačně identifikovatelná hodnotou klíče (případně hodnotami klíčů) seznamu. V mé práci jsem se setkal pouze se seznamy s jedním klíčem. Ve výpisu 2.1 je popsán model, jehož schéma definuje kontejner obsahující seznam s klíčem, který může nabývat hodnoty typu řetězec. Ve výpisu 2.2 se nachází YANG data zapsána ve formátu XML, jejichž schéma je popsáno ve výpisu 2.1. Ve výpisu 2.2 je možné si všimnout využití jmenného prostoru z popisu modelu 2.1 pro serializaci dat.

```
module bp-yang-priklad {
  yang-version 1.1;
  namespace "urn:janota:bp-yang-priklad";
  prefix bp;

  // toto je komentar
  container muj-kontejner {
    description
      "Kontejner zabalujici muj-seznam.";

    list muj-seznam {
      key "muj-list";
      description
        "Seznam s klicem 'muj-list'.";

      leaf muj-list {
        type string;
        description
          "List, ktery muze nabyvat hodnoty typu retezec.";
      } // muj-list
    } // muj-seznam
  } // muj-kontejner
} // bp-yang-priklad
```

Výpis 2.1: Ukázkový model popsaný v jazyku YANG.

```
<muj-kontejner xmlns="urn:janota:bp-yang-priklad">
  <muj-seznam>
    <muj-list>klic1</muj-list>
  </muj-seznam>
  <muj-seznam>
    <muj-list>klic2</muj-list>
  </muj-seznam>
</muj-kontejner>
```

Výpis 2.2: Příklad YANG dat ve formátu XML pro výpis 2.1.

Při implementaci jsem hojně využíval knihovnu *libyang*. *libyang* je *open-source* knihovna, která umí pracovat s YANG modely a daty. Knihovna umožňuje načíst YANG data například ve formátu XML a převést je do své interní reprezentace, ve které je pak možné s nimi dále pracovat. Tato data je mimo jiné možné validovat, což znamená zjistit, jestli přesně odpovídají popisu příslušného YANG modelu.

Zmíněné funkcionality knihovny by se neobešly bez tzv. *libyang* kontextu. Knihovna umožňuje zkompilovat YANG modely a nahrát je tak do kontextu. Za pomoci kontextu pak dokáže nad daty zkompilovaných modelů provádět operace. Při nahrávání modelu do kontextu je možné určit množinu vlastností tohoto modelu, které budou podporovány. Součástí projektu *libyang* je i program *yanglint*, který umožňuje procházet, tisknout a celkově

pracovat s YANG modely. Popisu toho, jak jsem s programem *yanglint* pracoval, a toho, jak jsem k němu přispěl, je věnována příloha A. Tento program jsem hojně využil při práci s *ietf-netconf-server* YANG modelem, o němž pojednává následující kapitola.

2.2 Protokol NETCONF

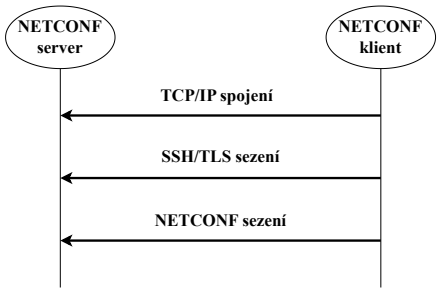
NETCONF (*Network Configuration Protocol*) je aplikační protokol definující mechanismus, kterým mohou být spravována síťová zařízení. Má široké využití v oblasti monitoringu a správy sítí. Byl standardizován organizací IETF a je mu přiděleno číslo portu 830 [7].

Protokol NETCONF využívá klient-server architekturu, kde server je obvykle nějaké síťové zařízení a klient je často nějaká aplikace. Protokol NETCONF vyžaduje spolehlivé spojení, avšak standard nespecifikuje konkrétní transportní protokol. Místo toho specifikuje požadavky na daný protokol, jimiž jsou autentizace, integrita dat, důvěrnost a zabezpečení proti přehrání. Z tohoto důvodu současně existují dva nejpoužívanější transportní protokoly, které se v rámci protokolu NETCONF nad TCP (*Transmission Control Protocol*) používají — SSH (*Secure Shell Protocol*) [12] a TLS (*Transport Layer Protocol*) [15].

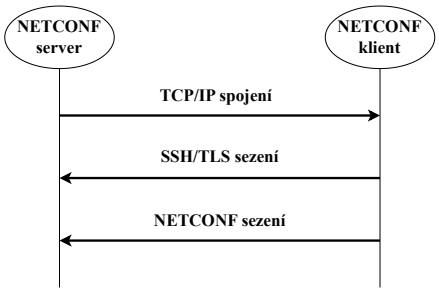
V protokolu NETCONF jsou data mezi klientem a serverem vyměňována pomocí tzv. RPC (*Remote Procedure Call*). RPC jsou při komunikaci zakódována do formátu XML. RPC umožňují zasílat operace (procedury), jejichž sémantika je popsána pomocí jazyka YANG. Mezi standardem definované operace patří například získání konfigurace ze serveru, změna či úprava této konfigurace a jiné.

Z hlediska uchování konfigurace byla v protokolu NETCONF definována architektura s názvem NMDA (*Network Management Datastore Architecture*), jejímž účelem je rozdělit data popsaná YANG modely podle konfigurace, kterou popisují. Tato architektura definuje datová úložiště různých typů, kterými jsou kandidátní (*candidate*), běžící (*running*) a startovací (*startup*) [3].

Z hlediska započetí komunikace mezi NETCONF klientem a serverem rozlišujeme dva módy, ve kterých server může operovat. Prvním módem je tzv. *listen* (naslouchací), kde komunikaci navazuje klient. Druhým typem je tzv. *Call Home* [20] (v doslovném překladu „volání domů“, dále už výhradně jako *Call Home*), což je mechanismus, kde komunikaci iniciuje server, nikoliv klient. Tento mód umožňuje například periodické získávání dat ze serveru, kdy je spojení periodicky navazováno a ukončováno ze strany serveru. Příklad navázání běžného *NETCONF* spojení je vyobrazen na obrázku 2.1 a příklad pro *Call Home* spojení na obrázku 2.2.



Obrázek 2.1: Průběh vytváření NETCONF spojení.



Obrázek 2.2: Průběh vytváření NETCONF *Call Home* spojení.

Důležitou součástí autentizace NETCONF klienta je získání jeho uživatelského jména a jeho práv. Tento proces je ponechán na použitém transportním protokolu. U protokolu SSH sděluje klient serveru své uživatelské jméno, které se pak použije i v rámci protokolu NETCONF [6]. Ovšem u protokolu TLS je tento proces složitější. Využívá se tzv. *cert-to-name*, což je mechanismus, který mapuje certifikáty na uživatelské jména [1]. Tento proces spočívá v tom, že si NETCONF server ukládá záznamy, které obsahují otisk (tj. hash celého certifikátu) a způsob získání uživatelského jména z certifikátu. Při autentizaci u protokolu TLS si server zažádá o klientský certifikát a porovná hodnotu otisku uloženou v tomto certifikátu s uloženými záznamy. Pokud se shodují, využije uloženou metodu pro zisk uživatelského jména. V případě, že uživatelské jméno nebylo získáno, nesmí být navázáno NETCONF spojení.

Aby se NETCONF klient mohl připojit na server, je žádoucí, aby server obsahoval alespoň jeden koncový bod. Koncový bod se dá chápat jako adresa v síti, kam klient může zasílat své požadavky a odkud na ně získá odpovědi. Koncový bod je na serveru identifikován svým jménem, v síti pak pomocí jeho síťové adresy a čísla portu. Po jeho vytvoření se na něj může NETCONF klient připojit a po úspěšné autentizaci dojde k vytvoření NETCONF sezení mezi danými zařízeními.

Pro úspěšné vytvoření NETCONF spojení si musí klient a server oba poslat tzv. *Hello* zprávu. Účelem těchto zpráv je domluvit se na verzi protokolu NETCONF, který se při komunikaci použije. Dalším smyslem je dohodnout se na „schopnostech“, kterými jsou mimo jiné současně myšleny i sady podporovaných vlastností YANG modelů, které byly popsány v sekci 2.1. Dále se pomocí těchto „schopností“ oba server i klient dozví, jaké RPC a operace jsou v rámci jejich komunikace podporovány.

Kapitola 3

Původní stav knihovny libnetconf2 a ietf-netconf-server YANG model

V této kapitole se zaměřím na detailnější popis knihovny *libnetconf2*, na její konfigurační API (*Application Programming Interface*), dále částečně na protokoly SSH a TLS a jejich použití v knihovně. Následně bude v této kapitole čtenář seznámen s modelem *ietf-netconf-server*. Na závěr zde porovnám konfigurační rozhraní knihovny s popisem modelu.

Knihovna *libnetconf2* je *open-source* projektem, který implementuje NETCONF klienta a server. Spojení mezi nimi je možno navázat buď pomocí protokolů SSH nebo TLS, anebo využitím lokální komunikace skrze unixovou schránku. Každý z těchto způsobů komunikace má své konfigurační možnosti.

3.1 Uživatelské rozhraní knihovny libnetconf2

Knihovna *libnetconf2* se snaží svým uživatelům poskytnout co nejpřívětivější a nejjednodušší uživatelské rozhraní. Původně knihovna *libnetconf2* obsahovala přes 50 různých API funkcí, které měly co do činění s konfigurací klienta nebo serveru. Nejprve začnu s popisem uživatelského rozhraní serveru, které je možno rozdělit do několika menších částí, a to:

- *server messages*,
- *server SSH*,
- *server session*,
- *server TLS*,
- *server-side call-home*,
- *ostatní*.

V části *server messages* se nachází API umožňující vytvoření, přizpůsobení a odeslání NETCONF zpráv zvaných RPC, viz sekce 2.2. Tato zpráva může sloužit jakožto odpověď po úspěšném dokončení operace, nebo naopak pokud nastala chyba. Rozhraní pro odpovídání na přijaté RPC je implementováno tak, že uživatel si sám definuje funkci pro každé serverem podporované RPC, kterou pomocí rozhraní serveru nastaví, a při obdržení daného RPC je tato uživatelsky definovaná funkce zavolána *libnetconf2* serverem. Chybové zprávy mohou

obsahovat několik atributů [7], které jim je možné nastavit. Mimo tyto RPC se v části *server messages* nachází API pro vytvoření upozornění (notifikací) [16] na nějaké události. Pokud žádaná událost nastane, tak server na to formou zprávy upozorní klienta. Tento mechanismus je založen na tom, že klient oznámí serveru, že chce přijímat dané upozornění o dané události.

V sekci *server SSH* se nachází rozhraní pro konfiguraci serveru, který umožňuje komunikaci pomocí protokolu SSH. Mezi nejdůležitější možnosti konfigurace z této sekce bych zařadil nastavení autentizačních metod. *libnetconf2* server umožňuje klienty autentizovat buď pomocí asymetrické kryptografie (tj. pomocí páru klíčů), hesla anebo interaktivně. Dále bych zde zařadil konfiguraci autorizovaných klíčů a hlavně privátního klíče serveru. Pro implementaci všeho okolo protokolu SSH využívá *libnetconf2 open-source* knihovnu *libssh*.

V části *server session* se nachází rozhraní pro správu NETCONF sezení ze strany serveru. Pro zjišťování stavu a získávání požadavků od klientů *libnetconf2* využívá tzv. *polling*, a tedy pasivně čeká na požadavky. Do této části patří například možnost vytvoření nových relací na všech koncových bodech serveru, zrušení těchto relací a vytvoření tzv. *pollsession*. *pollsession* je obálka, která umožňuje sdružit několik NETCONF sezení a provádět nad nimi již zmíněný *polling*.

Do oddílu *server TLS* jsou zařazeny možnosti konfigurace serveru, který poskytuje spojení pomocí protokolu TLS. Jedná se hlavně o nastavení certifikátu serveru a jeho certifikačních autorit, které slouží k ověření věrohodnosti klienty prezentovaných certifikátů. Další kritickou součástí konfigurace TLS serveru je nastavení tzv. *cert-to-name* záznamů, které slouží k rezoluci jména klienta, který chce navázat spojení (viz sekce 2.2). Pro implementaci protokolu TLS využívá *libnetconf2 open-source* knihovnu *OpenSSL*.

V předposlední části s názvem *server-side call-home* se nachází API pro práci s *Call Home* [20] NETCONF serverem. Aby server mohl navázat *Call Home* spojení s klienty dle figury 2.2, je potřeba, aby o nich věděl informace jako například jejich IP adresu. Proto se v této části nachází API, které umožňuje například nastavení informací o *Call Home* klientovi nebo délku periody opětovného připojení.

Ve skupině *ostatní* se nachází rozhraní, které je obecně nutné použít pro jakýkoliv *libnetconf2* server, nehledě na použitý transportní protokol nebo způsob komunikace. Mimo jiné se zde nachází API pro prvotní inicializaci serveru, vytvoření a konfiguraci koncových bodů a API pro ukončení serveru.

Uživatelské rozhraní klienta v knihovně *libnetconf2* je velice obdobné tomu u serveru. Dělí se na následující menší části:

- *client messages*,
- *client SSH*,
- *client session*,
- *client TLS*,
- *client-side call-home*,
- *ostatní*.

Podobně jako u serveru se v části *client messages* nachází API pro tvorbu a přizpůsobení RPC zpráv. Rozhraní poskytuje funkce pro tvorbu standardizovaných RPC, a tedy uživatel

knihovny si nemusí tyto zprávy tvořit sám. Jedním z těchto RPC je i již zmíněné oznámení o přijímání notifikací o dané události.

Ve skupině *client SSH* se vyskytuje uživatelské rozhraní pro konfiguraci chování klienta pro SSH spojení. Rozhraní poskytuje například nastavení asymetrického páru klíčů pro autentizaci, hesla a uživatelského jména klienta. Důležitou součástí této skupiny rozhraní je i nastavení „známých hostů“ (tj. *known hosts*), o kterých bude dále pojednávat kapitola 5. Dále se zde nachází API pro připojení se na server.

Následující část *client session* poskytuje API pro správu sezení. Do této části spadá hlavně API pro zaslání zprávy a získání odpovědi. Mimo jiné se zde nachází rozhraní pro získání „schnopností“ (viz sekce 2.2) prezentovaných serverem v úvodní výměně NETCONF zpráv mezi klientem a serverem.

V části *client TLS* se nachází rozhraní pro konfiguraci TLS klienta. Důležitou součástí této skupiny rozhraní je nastavení klientského certifikátu. Dále se zde nachází rozhraní pro konfiguraci certifikátů certifikačních autorit a seznamu zneplatněných certifikátů, jež slouží k ověření věrohodnosti certifikátu, jimž se prezentuje daný server.

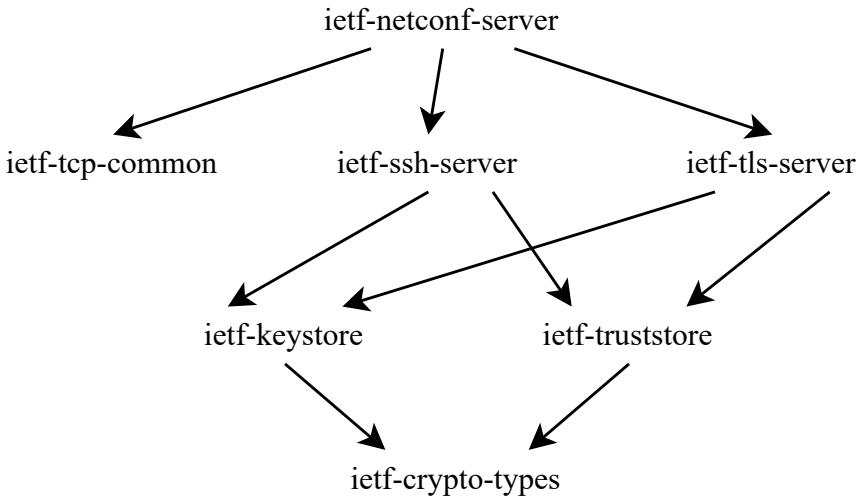
Předposlední část *client-side call-home* poskytuje API pro konfiguraci Call Home spojení ze strany klienta. Jedná se tedy o víceméně vše z kategorií *client SSH* a *client TLS*. Navíc se zde vyskytuje jedno API, které slouží k přijmutí spojení navázaného serverem.

A na závěr obdobně jako u serveru se ve skupině *ostatní* nachází API pro inicializaci a ukončení *libnetconf2* klienta. Mimo jiné se zde vyskytuje i API pro získávání YANG modelů, což je možné provést například pomocí nastavení adresáře, ve kterém se dané YANG modely vyskytují.

3.2 ietf-netconf-server YANG model

ietf-netconf-server je YANG model popisující konfigurační možnosti NETCONF serveru [21]. V době psaní této práce ještě tento model nebyl standardizovaný, a tudíž se jedná pouze o návrh. V budoucnu by však měl být standardizován a jeho budoucí verze se může trochu lišit od té popsané a odkazované v této práci. I přesto však v této práci používám slovo „standardizovaný“, i když současně je to pouze „návrh standardu“. Představení modelu bude v této sekci provedeno primárně pomocí stromových diagramů, jejichž účelem je popsat strukturu modelu. Tyto diagramy jsou doplněny o slovní popis významu některých uzlů. K práci s tímto YANG modelem jsem využil hlavně program *yanglint*, který je součástí projektu *libyang* a jemuž je věnována příloha A. Při popisu modelu *ietf-netconf-server* uvádím pouze některé podstromy a uzly, které jsou závislé na podporování nějaké YANG vlastnosti (opět viz sekce 2.1).

Model má několik závislostí, tedy jiných YANG modelů, které importuje. Zjednodušené schéma závislostí je možné vidět na obrázku 3.1. Avšak nejpodstatnější z importovaných modelů jsou v tomto textu pouze dva — *ietf-keystore* [22] a *ietf-truststore* [23]. Tyto modely definují postupně tzv. *keystore* a *truststore* úložiště, která slouží k uložení citlivých údajů, kterými jsou zejména *Base64* [9] zakódované certifikáty, symetrické a asymetrické klíče. Dále v textu se pak při popisu *ietf-netconf-server* YANG modelu často vyskytuje volba mezi lokálním uložením (tj. uzel *inline-definition*) a referencí do jednoho z těchto dvou úložišť (tj. například uzel *central-keystore-reference*). Tato volba umožňuje buď citlivá data definovat přímo v YANG datech daného modelu, anebo zde předat odkaz do úložiště, kde se tato data vyskytují.



Obrázek 3.1: Zjednodušené schéma závislostí YANG modelu *ietf-netconf-server*.

Model *ietf-netconf-server* popisuje konfiguraci jak naslouchacího, tak i *Call Home* módu, což je znázorněno diagramem 3.1. Současně oba tyto módy obsahují dva podstromy, které udávají popis pro spojení pomocí protoklů SSH (*Secure Shell Protocol*) a TLS (*Transport Layer Protocol*).

```
netconf-server
├── listen
│   └── ...
└── call-home
    └── ...
```

Diagram 3.1: Struktura kontejneru *netconf-server*.

3.2.1 Konfigurace módu poslouchání pro nová spojení

Konfigurace serveru, který poslouchá pro nová spojení, je popsána v kontejneru *listen* a znázorněna na diagramu 3.2. Jeho prvním potomkem je list *idle-timeout*, který udává maximální čas v sekundách, po který může být NETCONF relace nečinná. Po vypršení tohoto času uzavírá NETCONF server spojení. Druhou položkou je kontejner *endpoints*, který zabaluje seznam koncových bodů, na které se klienti mohou připojit. Identifikátorem záznamu tohoto seznamu je unikátní název. Seznamy jsou v diagramech naznačeny symbolem „*“, po němž následuje název klíče v hranatých závorkách. Koncový bod může umožňovat buď SSH, nebo TLS spojení.

Konfigurace SSH u módu pro poslouchání

Kontejner *ssh* umožňuje konfiguraci SSH serveru. Jeho struktura je vyobrazena na diagramu 3.3. Jeho prvním potomkem je kontejner *tcp-server-parameters*, jehož významem je nakonfigurování adresy a portu koncového bodu. Dále obsahuje *presence* kontejner (znázorněno v diagramu 3.3 symbolem „!“) s názvem *keepalives*. Tento kontejner a jeho potomci slouží k povolení a správě periodických zpráv, jejichž účelem je periodicky informovat dru-

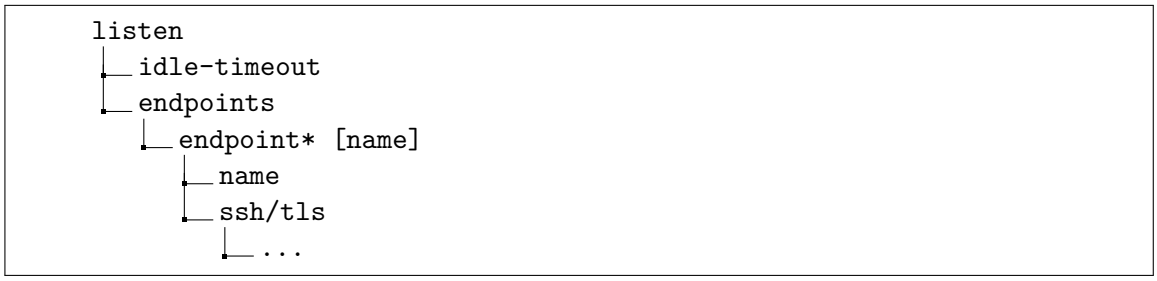


Diagram 3.2: Struktura kontejneru *listen*.

hou stranu o tom, že spojení je stále aktivní. Tyto *keepalive* zprávy umožňují například zabránit automatickému ukončení relace z důvodu nečinnosti. Záměrně prvně přeskočím kontejner *ssh-server-parameters* a nejprve popíši dalšího přímého potomka *ssh*, jímž je kontejner *netconf-server-parameters*. Potomci tohoto kontejneru umožňují nakonfigurovat *cert-to-name* záznamy pro využití certifikátů při autentizaci u protokolu SSH, avšak existence tohoto kontejneru je podmíněna podporováním dané vlastnosti (popsáno v sekci 2.1). A tedy posledním potomkem kontejneru *ssh* je podstrom *ssh-server-parameters*, který se pouze dělí na čtyři dílčí kontejnery.

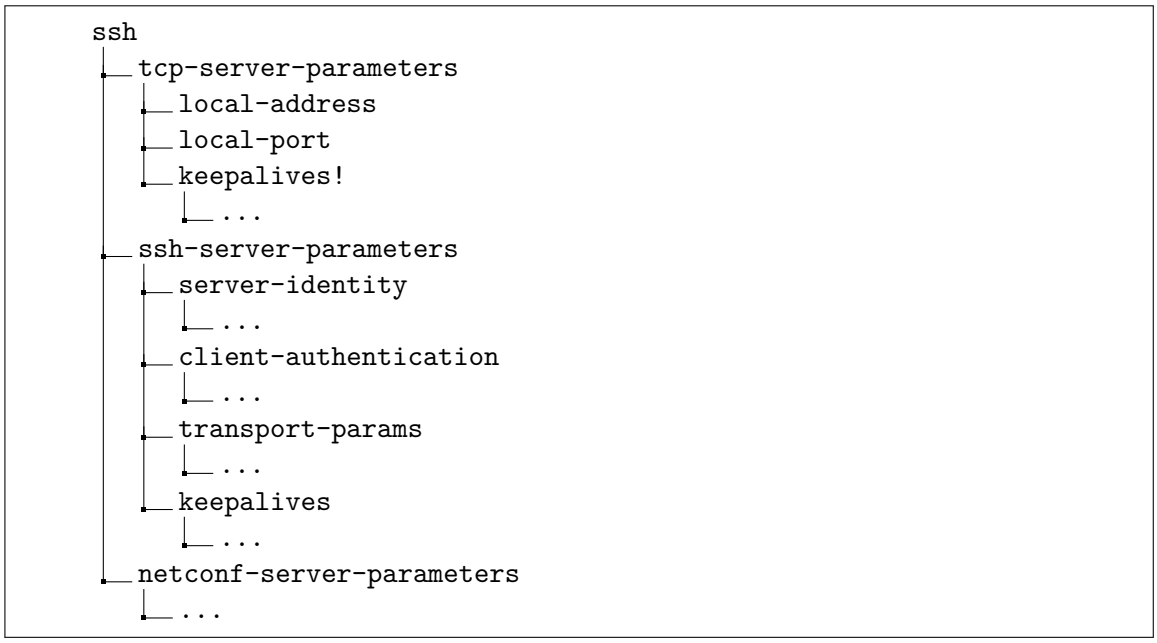


Diagram 3.3: Struktura kontejneru *ssh*.

Prvním potomkem v podstromu *ssh-server-parameters* je kontejner *server-identity* znázorněný diagramem 3.4, který popisuje identitu SSH serveru. Obsahuje seznam *host-key*, jehož identifikátorem je unikátní název. Je to seznam buď certifikátů, anebo párů veřejných a privátních klíčů, jejich typů a hodnot, které plní funkci tzv. *SSH host key* [12]. Tyto certifikáty anebo páry klíčů mohou být uloženy buď lokálně přímo v YANG datech *ietf-netconf-server* modelu, anebo tato data mohou obsahovat pouze odkaz do úložiště *keystore*.

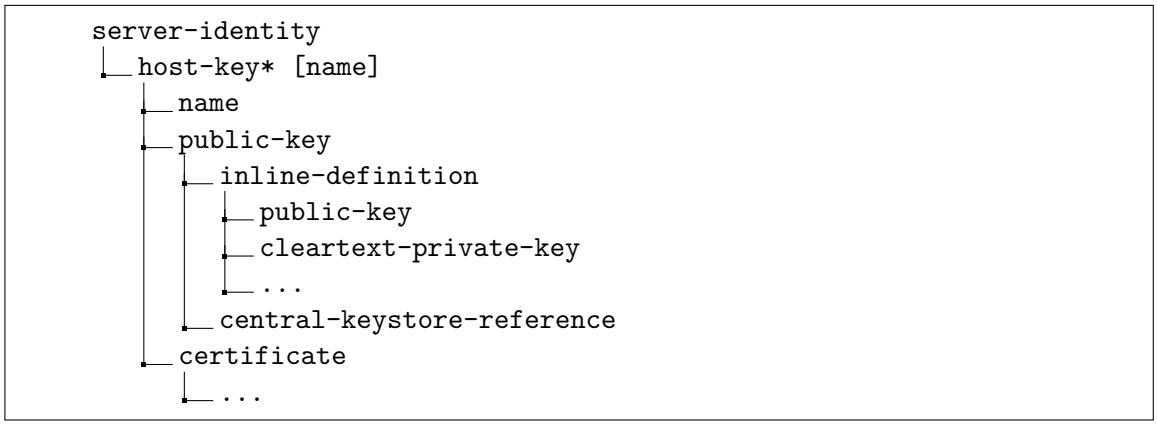


Diagram 3.4: Struktura kontejneru *server-identity* pro podstrom *ssh*.

Dalším potomkem uzlu *ssh-server-parameters* je kontejner *client-authentication* popsaný diagramem 3.5, který nese informaci o uživatelích, kteří se mohou na daném koncovém bodu autentizovat pomocí protokolu SSH. Záznam v seznamu *user* reprezentuje jednoho uživatele, který je identifikovaný svým jménem. V tomto seznamu je možné pro každého uživatele nakonfigurovat jeho vlastní způsoby autentizace, avšak uživatel se musí autentizovat za pomoci všech jeho nastavených metod. Dále je zde možné nakonfigurovat certifikáty pro autentizaci pomocí protokolu SSH, nicméně uzly týkající se této konfigurace jsou podmíněné podporováním dané vlastnosti.

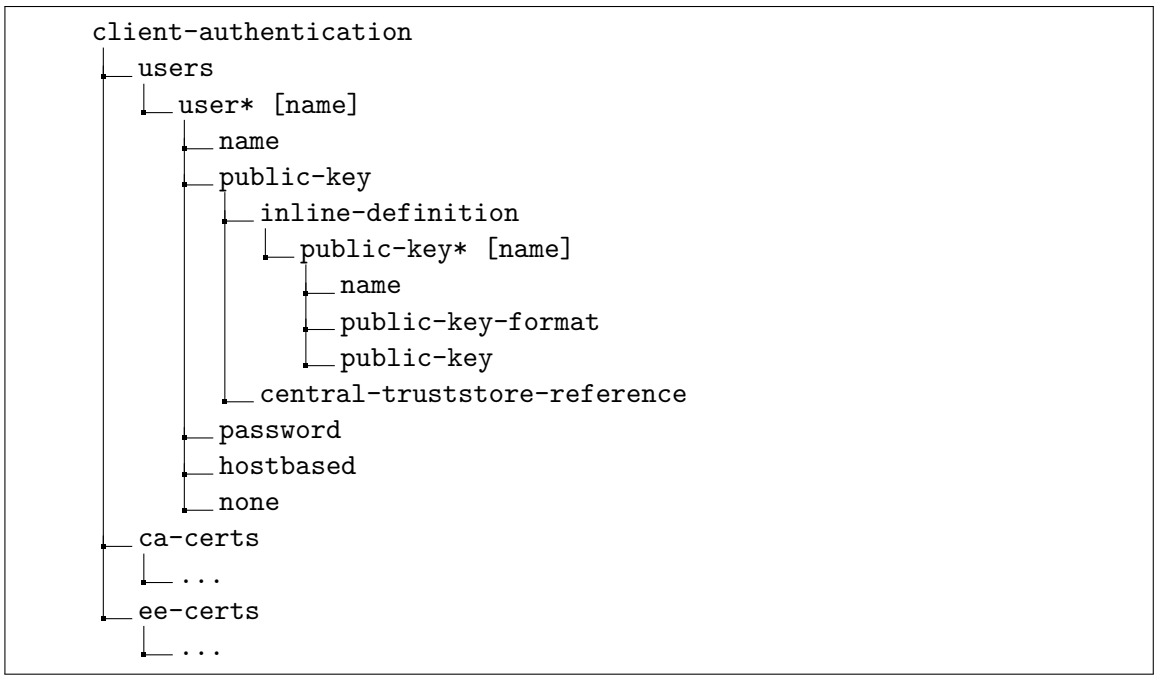


Diagram 3.5: Struktura kontejneru *client-authentication* pro podstrom *ssh*.

Předposledním potomkem uzlu *ssh-server-parameters* je kontejner s názvem *transport-params*, jehož potomci umožňují konfiguraci parametrů a celkově zabezpečení SSH komunikace [12]. Mezi tyto parametry patří podporované algoritmy veřejných klíčů (např. *RSA*), algoritmy pro výměnu veřejných klíčů (např. *Diffie-Hellman*), algoritmy pro šifrování dat

(např. *AES*) a MAC (*Message Authentication Code*) algoritmy (např. *MD5*). Posledním potomkem uzlu *ssh-server-parameters* je kontejner *keepalives*, jehož význam je stejný jako v kontejneru *tcp-server-parameters*, avšak tyto *keepalive* zprávy se vztahují pouze na protokol SSH.

Konfigurace TLS u módu pro poslouchání

Potomci kontejneru *tls* umožňují konfiguraci TLS serveru. Struktura tohoto kontejneru je zobrazena na diagramu 3.6. Obsahuje tři přímé potomky, kterými jsou kontejnery *tcp-server-parameters*, *tls-server-parameters* a *netconf-server-parameters*, z toho první z nich je naprosto identický s tím u protokolu SSH.

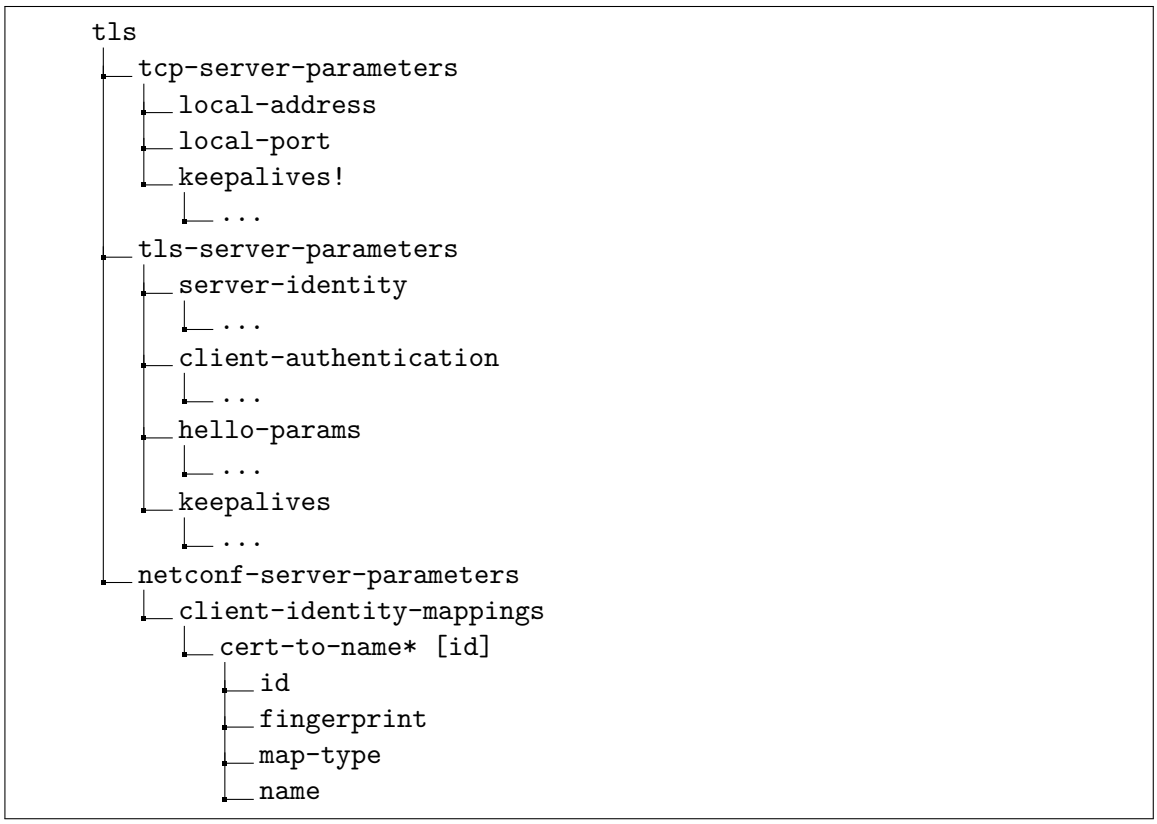


Diagram 3.6: Struktura kontejneru *tls*.

Kontejner *server-identity*, jehož strukturu je možné vidět na diagramu 3.7, umožňuje nakonfigurovat identitu TLS serveru. První možností je nastavení certifikátu, kterým se server prezentuje, a to buď lokálně, nebo odkazem do úložiště. Druhým způsobem, kterým se NETCONF TLS server může prezentovat, je pomocí čistě privátního klíče [24] obdobně jako u SSH. Třetí a čtvrtý způsob je založený na využití předsdílených klíčů (tzv. *pre-shared keys* nebo také *pairwise-symmetric keys*), a to za pomoci uzlů v podstromech *tls12-psk* [17] a *tls13-epsk* [15]. Všechny zmíněné kontejnery jsou založené na podporování dané vlastnosti.

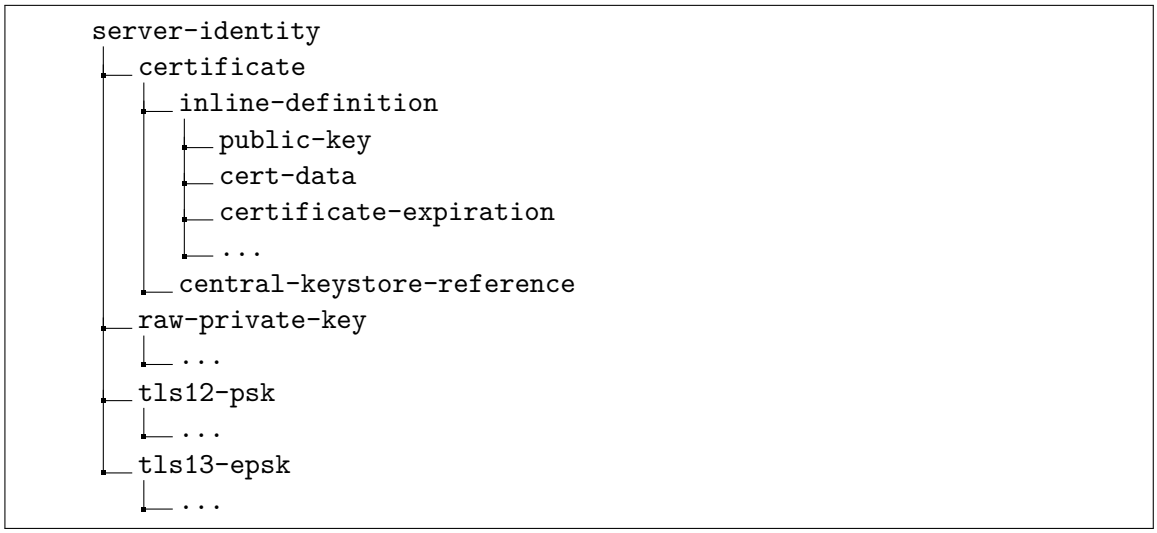


Diagram 3.7: Struktura kontejneru *server-identity* pro podstrom *tls*.

Následuje kontejner *client-authentication* znázorněný na diagramu 3.8. Potomci tohoto kontejneru slouží k uložení mechanismů, pomocí nichž se mohou klienti autentizovat. Jeho prvním potomkem je kontejner *ca-certs* (*Certificate Authority certificates*), který zabaluje skupinu certifikátů certifikačních autorit. Klient je autentizován, pokud je možné z certifikátu, kterým se prezentuje, vytvořit validní „přenos důvěry“ (tj. *chain of trust*) k alespoň jednomu z nakonfigurovaných certifikátů certifikačních autorit. Dalším potomkem je kontejner *ee-certs* (*End Entity certificates*), který má stejnou strukturu jako kontejner *ca-certs* a který definuje možnosti konfigurace klientských certifikátů. Klient je autentizován, pokud se jeho certifikát shoduje s jedním z nakonfigurovaných. Kontejner *raw-public-keys* slouží k autentizaci pomocí prostých veřejných klíčů podobně jako u SSH. A na konec se zde nachází dva listy, kterými jsou *tls12-psks* a *tls13-epsks*, jejichž přítomnost indikuje, že daný koncový bod podporuje autentizaci klientů za pomoci těchto mechanismů.

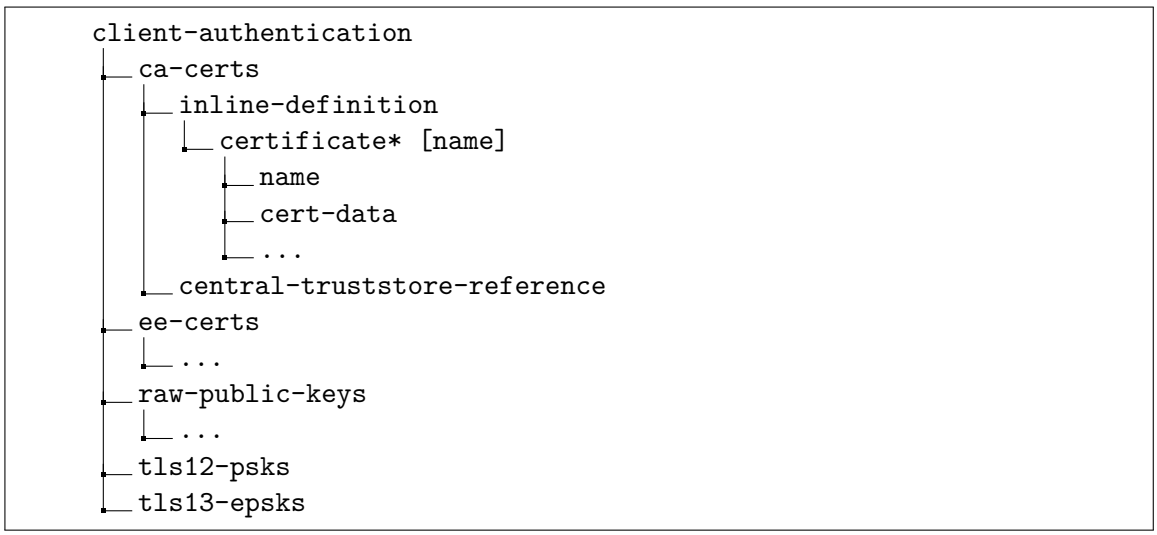


Diagram 3.8: Struktura kontejneru *client-authentication* pro podstrom *tls*.

Předposledním potomkem v podstromu *tls-server-parameters* je kontejner *hello-params*, který definuje uzly pro konfiguraci parametrů tzv. *ServerHello* [15] zprávy protokolu TLS. Těmito parametry jsou dva seznamy — seznam podporovaných verzí protokolu TLS a seznam podporovaných sad šifer. Posledním potomkem je kontejner *keepalives*, který má obdobný význam jako u kontejneru *tcp-server-parameters*, avšak vztahuje se pouze na protokol TLS[1].

Na závěr posledním potomkem podstromu *tls* je kontejner *netconf-server-parameters*, který obsahuje seznam mapování uživatelských jmen na certifikáty. Tento proces rezoluce uživatelského jména je popsán v sekci 2.2.

3.2.2 Konfigurace módu Call Home

Call Home mód popsaný v modelu *ietf-netconf-server* je podobný tomu naslouchacímu, jak lze vidět na diagramu 3.9. Navíc podstrom *call-home* definuje seznam s názvem *netconf-client*, který představuje jednoho NETCONF klienta. Server se podle nastavených parametrů pokouší navázat spojení s klientem na jeho koncových bodech.

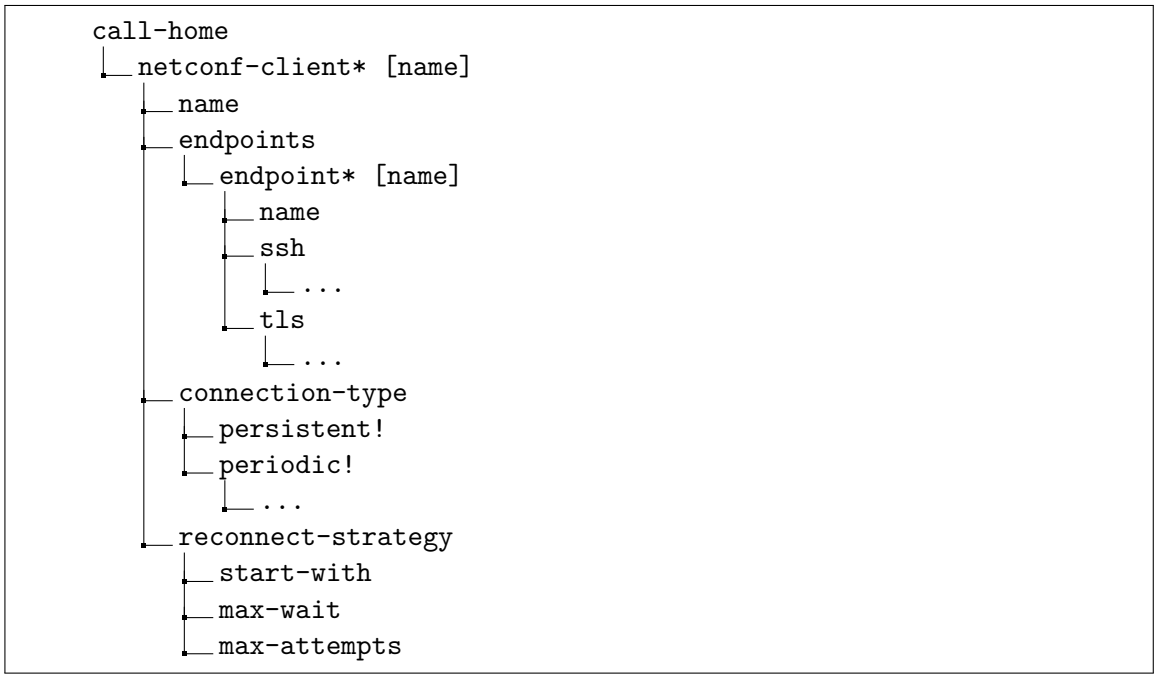


Diagram 3.9: Struktura kontejneru *call-home*.

Typ tohoto spojení je možné nakonfigurovat pomocí uzlů podstromu *connection-type*. Rozlišujeme dva druhy spojení — persistentní a periodické. Persistentní spojení je navázáno nepřetržitě a pokud by došlo k jeho ztrátě, musí být okamžitě zahájeny pokusy o jeho obnovení, a to na základě parametrů nastavených v kontejneru *reconnect-strategy*. Naopak u periodického typu spojení se server pravidelně pokouší připojit na klienta. Po dokončení plánovaných aktivit by měl klient ukončit spojení. Periodicita těchto spojení a další parametry je možné nakonfigurovat pomocí potomků *presence* kontejneru *periodic*. Kontejner *reconnect-strategy* definuje uzly pro způsob znovunavázání spojení. Obvykle to funguje tak, že při ztrátě spojení s klientem server vybere koncový bod dle hodnoty listu *start-with*.

[1] Tento mechanismus v minulosti způsobil bezpečnostní riziko s názvem „Heartbleed“.

Server se pokusí navázat spojení na tomto koncovém bodu. Při dosažení *max-attempts* neúspěchů se zvolí další koncový bod. List *max-wait* definuje maximální dobu, po které je pokus o spojení prohlášen za neúspěšný.

Dalším rozdílem oproti naslouchacímu módu je ten, že kontejnery *tcp-server-parameters* byly nahrazeny za *tcp-client-parameters*, které obsahují jiný popis, viz diagram 3.10. Potomci tohoto kontejneru umožňují nakonfigurovat síťovou adresu a port klienta, na kterého se server pokusí připojit (tj. *remote*). Dále umožňují nastavit adresu a port, ze kterých má server navázat spojení (tj. *local*). Dále pomocí *presence* kontejneru *proxy-server* model umožňuje nakonfigurovat proxy server pro navázání *Call Home* spojení.

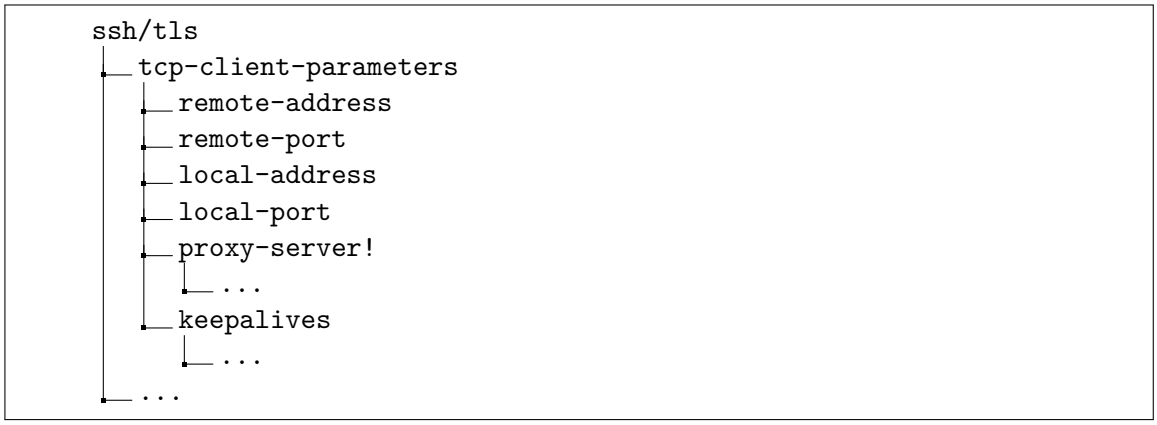


Diagram 3.10: Struktura kontejneru *tcp-client-parameters*.

3.3 Porovnání konfiguračního rozhraní knihovny libnetconf2 s modelem ietf-netconf-server

V této sekci bude porovnáno konfigurační API knihovny *libnetconf2* s popisem daným YANG modelem *ietf-netconf-server*. Porovnání je představeno formou tabulek, které mají dva sloupce. V levém horním rohu je vždy název zkoumaného podstromu. Symbol „✓“ nacházející se v pravém sloupci značí, že konfigurace daného uzlu nebo podstromu je v původním API možná. Výskyt symbolu „✗“ v tomto sloupci značí absenci možnosti konfigurace daného uzlu nebo podstromu.

listen	Je v API
idle-timeout	✓
endpoint	✓

Tabulka 3.1: Porovnání uzlů kontejneru *listen* s konfiguračním rozhraním knihovny.

endpoint	Je v API
ssh	✓
tls	✓

Tabulka 3.2: Porovnání uzlů seznamu *endpoint* s konfiguračním rozhraním knihovny.

Z tabulek 3.1 a 3.2 je patrné, že konfigurační API knihovny *libnetconf2* umožňuje nakonfigurovat zmíněné uzly modelu. V tabulce 3.2 oproti schéma 3.2 chybí uzel s názvem *name*, avšak tento uzel je klíčem seznamu *endpoint*, který je podporován, tudíž implicitně je možné nakonfigurovat i tento list. Dále už budou klíče podporovaných seznamů myšleny jako podporovány implicitně.

3.3.1 Porovnání SSH

Porovnání uzlů prvního potomka kontejneru *ssh*, tedy kontejneru *tcp-server-parameters*, je možné vidět v tabulce 3.3. Všechny jeho potomky je možné pomocí knihovny *libnetconf2* nakonfigurovat.

tcp-server-parameters	Je v API
local-address	✓
local-port	✓
keepalives	✓

Tabulka 3.3: Porovnání uzlů SSH kontejneru *tcp-server-parameters* s konfiguračním rozhraním knihovny.

Následuje porovnání kontejneru *server-identity*, které je popsané tabulkou 3.4. Knihovna *libnetconf2* obsahuje API pro konfiguraci tzv. *SSH host key* [12], nicméně pouze za pomoci asymetrického páru klíčů. Certifikáty tímto způsobem není možné nakonfigurovat, jelikož knihovna *libnetconf2* pro implementaci protokolu SSH využívá knihovnu *libssh*, ve které chybí tato funkcionalita.

Porovnání v tabulce 3.4 není úplně přesné, jelikož model *ietf-netconf-server* definuje několik způsobů, jak je možné lokálně uložit privátní klíč v podstromu *inline-definition*. Z těchto způsobů knihovna podporuje pouze konfiguraci privátního klíče v nezašifrované podobě (tj. *cleartext private key*). Nicméně tento rozdíl nepovažuji za zásadní.

Ovšem velkým rozdílem je způsob, jakým je v knihovně získáván tento *SSH host key*. Konfigurační rozhraní knihovny ponechává kompletně na uživatelovi způsob obdržení privátního klíče, a to ať už například načtením ze souboru nebo předáním přímo *Base64* zakódovaných dat. Při autentizaci je vždy pomocí tohoto uživatelem definovaného mechanismu získán privátní klíč, ze kterého se vygeneruje veřejný klíč a který se použije v komunikaci s klientem. Naopak model *ietf-netconf-server* striktně vyžaduje, aby se privátní a veřejný[2] klíč nacházel v YANG datech.

server-identity	Je v API
host-key	✓
inline-definition	✓
central-keystore-reference	✓
certificate	✗

Tabulka 3.4: Porovnání uzlů SSH kontejneru *server-identity* s konfiguračním rozhraním knihovny.

V kontejneru *client-authentication* dle tabulky 3.5 není možné pomocí API knihovny nakonfigurovat nic. Certifikáty není možné nakonfigurovat ze stejného důvodu, jako tomu bylo u kontejneru *server-identity*. Nastavení uživatelů, kteří mohou být úspěšně autentizováni, funguje v knihovně oproti modelu odlišně. Seznam autorizovaných SSH klientů je v knihovně získáván ze systému, na kterém je *libnetconf2* server spuštěn. Tedy jinými slovy knihovna neobsahuje API na definici autorizovaných klientů. Dalším rozdílem je, že model definuje autentizační metody pro každého uživatele zvlášť. V knihovně je možné tyto

[2] Veřejný klíč byl původně povinnou součástí konfigurace *SSH host key*, avšak čtenář se dále v práci dozví, proč už tomu tak není.

metody nakonfigurovat pouze globálně pro všechny uživatele. Model dále vyžaduje, že se klient musí autentizovat pomocí všech jeho nastavených metod, avšak v knihovně stačí, aby se úspěšně autentizoval pouze pomocí jedné z nich.

client-authentication	Je v API
users	✗
ca-certs	✗
ee-certs	✗

Tabulka 3.5: Porovnání uzlů SSH kontejneru *client-authentication* s konfiguračním rozhraním knihovny.

Poslední dva potomky uzlu *ssh-server-parameters* dle diagramu 3.3, tedy kontejnery *transport-params* a *keepalives* není možné v knihovně nakonfigurovat. To stejné platí pro kontejner *netconf-server-parameters*, jelikož není potřeba provádět rezoluci certifikátu na uživatelské jméno, pokud certifikáty v knihovně nejsou u protokolu SSH podporovány.

3.3.2 Porovnání TLS

U porovnání podstromu *tls* s API knihovny budu vycházet z diagramu 3.6. Jeho prvním potomkem je kontejner *tcp-server-parameters*, pro který platí to stejné jako u transportního protokolu SSH a současně tedy i tabulka 3.3.

Dále se budu věnovat kontejneru *server-identity*, jehož porovnání popisuje tabulka 3.6. Knihovna *libnetconf2* umožňuje nakonfigurovat certifikát serveru stejným mechanismem, jako tomu je s *SSH host key* u protokolu SSH. Avšak ne všechny uzly podstromu *certificate* jsou podporovány. Příkladem uzlu, který není možné nakonfigurovat, je notifikace o vypršení platnosti certifikátu. Nicméně tento rozdíl nepovažuji v rámci této práce za podstatný. Zbývající podstromy dle tabulky 3.6 není možné nakonfigurovat.

server-identity	Je v API
certificate	✓
raw-private-key	✗
tls12-psk	✗
tls13-epsk	✗

Tabulka 3.6: Porovnání uzlů TLS kontejneru *server-identity* s konfiguračním rozhraním knihovny.

Následuje porovnání kontejneru *client-authentication*. Dle tabulky 3.7 je v knihovně možné nakonfigurovat jak certifikáty klientské, tak i certifikačních autorit. Knihovna ale nepodporuje autentizaci pomocí párů klíčů bez certifikátů při použití transportního protokolu TLS. Mechanismy autentizace popsané uzly *tls12-psks* a *tls13-epsks* nejsou v knihovně dostupné. Možným důvodem je jejich absence ve starších verzích modelu *ietf-netconf-server*, jejichž popisem bylo původní konfigurační rozhraní knihovny inspirováno.

Dle diagramu 3.6 obsahuje podstrom *tls-server-parameters* ještě dva potomky, kterými jsou kontejnery *hello-params* a *keepalives*. Žádný z nich není v knihovně podporován.

Poslední zkoumaný podstrom má název *netconf-server-parameters*. Porovnání s API knihovny je znázorněno tabulkou 3.8. Knihovna *libnetconf2* podporuje proces *cert-to-name* (viz sekce 2.2), tudíž všechny tyto uzly je možné nakonfigurovat.

client-authentication	Je v API
ca-certs	✓
ee-certs	✓
raw-public-keys	✗
tls12-psks	✗
tls13-epsks	✗

Tabulka 3.7: Porovnání uzlů TLS kontejneru *client-authentication* s konfiguračním rozhraním knihovny.

netconf-server-parameters	Je v API
id	✓
fingerprint	✓
map-type	✓
name	✓

Tabulka 3.8: Porovnání uzlů kontejneru *netconf-server-parameters* s konfiguračním rozhraním knihovny.

3.3.3 Porovnání módu Call Home

Porovnání konfiguračního rozhraní knihovny pro *Call Home* mód s popisem *ietf-netconf-server* modelu je částečně popsán tabulkou 3.9. API knihovny umožňuje nakonfigurovat koncové body zvlášť pro každého *Call Home* klienta. Současně dovoluje nastavit jak periodické, tak persistentní spojení. U strategie zpětného připojení chybí možnost nakonfigurovat uzel *max-wait* (viz 3.2.2), avšak ostatní možnosti spolu s funkcionalitou se zde nachází.

netconf-client	Je v API
endpoints	✓
connection-type	✓
reconnect-strategy	✓

Tabulka 3.9: Porovnání uzlů *Call Home* kontejneru *netconf-client* s konfiguračním rozhraním knihovny.

Jelikož podstromy *ssh* a *tls* obou módů jsou v modelu *ietf-netconf-server* téměř totožné, tak zbývá porovnat pouze obsah kontejneru *tcp-client-parameters*. Porovnání je znázorněno tabulkou 3.10. Knihovna *libnetconf2* obsahuje API pro konfiguraci síťové adresy a portu vzdáleného klienta, avšak neumožňuje nastavit tyto parametry pro rozhraní, odkud se má spojení navázat. Knihovna dále nepodporuje konfiguraci proxy serveru. Nastavení *keepalives* je totožné s tím u módu pro poslouchání.

tcp-client-parameters	Je v API
remote-address	✓
remote-port	✓
local-address	✗
local-port	✗
proxy-server	✗
keepalives	✓

Tabulka 3.10: Porovnání uzlů *Call Home* kontejneru *tcp-client-parameters* s konfiguračním rozhraním knihovny.

3.4 Shrnutí porovnání

Při porovnání YANG modelu *ietf-netconf-server* s konfiguračním rozhraní knihovny *libnetconf2* jsem zjistil, že největší rozdíl se nachází v absenci možnosti nakonfigurovat si uživatele, kteří se mohou pomocí protokolu SSH autentizovat. Tito uživatelé jsou získávání ze systému, na kterém je *libnetconf2* NETCONF server spuštěn. Pro jejich konfiguraci neexistuje žádné API.

Dalším podstatným rozdílem je mechanismus, kterým jsou knihovnou získávány asymetrické klíče a certifikáty. Klíče ani certifikáty není možné nakonfigurovat přímo pomocí rozhraní. V knihovně *libnetconf2* to funguje tak, že uživatel nastaví mechanismus, pomocí nějž jsou klíče a certifikáty získávány. Knihovna následně využije daný nastavený mechanismus až v ten moment, kdy je daný klíč nebo certifikát potřeba.

Mezi další rozdíly patří konfigurace certifikátů v rámci protokolu SSH. Tato možnost v knihovně chybí. Dalším rozdílem je, že identitu TLS serveru je možné reprezentovat pouze pomocí certifikátu, ale nikoliv za pomoci jiných mechanismů popsaných v modelu *ietf-netconf-server*. To stejné platí i pro autentizaci u protokolu TLS. Co se *Call Home* módu týče, tak pro něj platí vše výše popsané. Navíc zde mimo jiné chybí možnost konfigurace proxy serveru a rozhraní, odkud se má server pokoušet o připojení se na klienta (tj. *local-address* a *local-port*).

Kapitola 4

Návrh nového konfiguračního rozhraní knihovny libnetconf2

Tato kapitola je věnována návrhu nového konfiguračního rozhraní knihovny. Návrh vychází z porovnání, které je popsáno v kapitole 3. Oproti stávajícímu API, které je založeno na konfiguraci serveru po dílčích částech, jsem přišel s návrhem rozhraní, které nakonfiguruje celý server do cílové podoby za pomoci de facto jednoho volání funkce poskytované rozhraním knihovny.

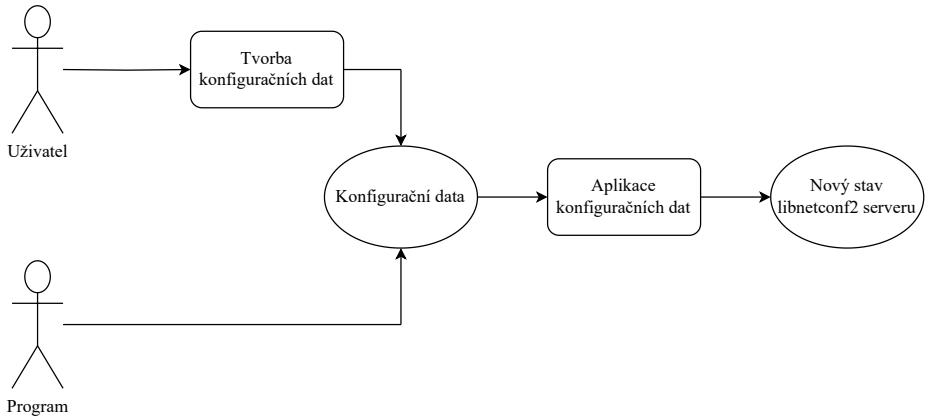
Původní konfigurační rozhraní knihovny bylo založeno na postupné konfiguraci po dílčích částech. To vedlo na spoustu různých funkcí rozhraní, rozsáhlou dokumentaci a delší kód programů, které využívaly knihovnu *libnetconf2*. Dalším problémem rozhraní při použití protokolu SSH byla absence možnosti nakonfigurovat uživatele, kteří se mohou na server připojit. Tito uživatelé se získávali ze systému, na kterém byla knihovna spuštěna.

Návrh nového rozhraní je postaven na tom, že knihovna *libnetconf2* musí být kompatibilní s alespoň nějakou podmnožinou *ietf-netconf-server* YANG modelu. To znamená, že se zvolí nějaká množina vlastností (viz sekce 2.1) modelu *ietf-netconf-server* a modelů, které importuje. Tato množina vlastností bude knihovnou podporována. Pak všechny uzly, jejichž definice je založena na podporování pouze vybraných vlastností, musí být konfigurovatelné knihovnou. Jako příklad uvažujme YANG vlastnost s názvem *ssh-x509-certs*, která je součástí YANG modelu *ietf-ssh-common* a která reprezentuje použití certifikátů v rámci transportního protokolu SSH. Pokud bych si při implementaci zvolil podporovat tuto vlastnost, bylo by nutné implementovat konfiguraci a celkově použití těchto certifikátů v rámci protokolu SSH. Naopak, pokud bych se rozhodl tuto vlastnost nepodporovat, tak je možné uzly závislé na této vlastnosti ignorovat, a i přesto zachovat kompatibilitu s popisem *ietf-netconf-server* YANG modelu.

Vstupem nového konfiguračního rozhraní pak mohou být YANG data *ietf-netconf-server* (případně *ietf-keystore* nebo *ietf-truststore*) modelu. Pomocí později představeného algoritmu dojde k zpracování konfiguračních YANG dat a na výstupu bude *libnetconf2* server, který je nakonfigurován dle vstupu.

Nové rozhraní je popsané na obrázku 4.1. V mém návrhu odlišuji dva aktéry, které jsem nazval „uživatel“ a „program“. Rozdíl mezi nimi je takový, že program si konfigurační data zajistí vlastní cestou a nepotřebuje k tomu API knihovny. Pro běžného uživatele pak konfigurace serveru představuje dva kroky, kterými jsou tvorba a následně aplikace konfiguračních dat.

Rozdílem mezi původním a novým rozhraním je rozhodně počet volání API, který je potřebný ke konfiguraci serveru. Velkou výhodou nového způsobu je, že si uživatel může konfigurační YANG data exportovat například do formátu XML a následně je komukoliv jednoduše sdílet. Další výhodou je jednotný formát konfigurace, který je zaručen kompatibilitou s *ietf-netconf-server* modelem.



Obrázek 4.1: Návrh nového konfiguračního rozhraní knihovny.

4.1 Návrh rozhraní pro aplikaci konfiguračních dat

Pro aplikaci konfiguračních YANG dat jsem navrhl algoritmus, který prochází strom s daty do hloubky. Jeho podstatou je navštívit každý uzel a podle jeho názvu provést danou konfiguraci. Avšak otázkou bylo, co dělat s předchozí konfigurací serveru. Proto jsem navrhl dva způsoby aplikace konfiguračních dat, které závisí na atributu „operace“. Pro představu například u YANG dat ve formátu XML je slovy „atribut operace“ myšlen doslovně atribut nějakého uzlu. Tyto operace jsou inspirované standardizovaným RPC s názvem *edit-config* [7]. Atribut operace může nabývat čtyř hodnot, kterými jsou:

- *create* - vytvořit,
- *replace* - nahradit,
- *delete* - smazat,
- *none* - nic.

První navržený způsob aplikace konfiguračních dat je založený na tom, že alespoň kořenový uzel YANG dat obsahuje atribut operace. Jeho potomci buď definují novou hodnotu tohoto atributu, nebo ji zdědí. Nová konfigurace pak závisí na hodnotách těchto atributů.

Druhý způsob aplikace konfiguračních dat je založen na tom, že nezáleží na předchozí konfiguraci. To znamená, že veškerá předchozí konfigurace se zahodí a nahradí se novou (tj. jakoby všechny uzly měly atribut operace s hodnotou „nahradit“). Žádný uzel YANG dat přitom nesmí obsahovat atribut operace.

První způsob umožňuje větší volnost, co se konfigurace týče, jelikož dokáže přesně vyjádřit změny oproti předchozí konfiguraci. Nicméně je o to víc pracné vytvořit daná YANG data, která budou vstupem algoritmu, a současně roste riziko nějaké chyby nebo chybné

konfigurace serveru. Dalším podstatným rozdílem je, že pro použití prvního způsobu si uživatel musí uchovávat předchozí konfiguraci, jelikož její části mohou být zachovány vzhledem k použitým atributům operace.

Předpokladem úspěšné aplikace konfiguračních dat je, že YANG data, ať už *ietf-netconf-server*, *ietf-keystore* nebo *ietf-truststore* modelu, musí být validní dle standardu [2]. Podle mého návrhu je validace dat ponechána na uživateli, a tedy musí ji sám zajistit před konfigurací *libnetconf2* serveru. Validace je ponechána na uživateli z toho důvodu, že kdyby už například měl danou validní konfiguraci uloženou, tak by nové rozhraní provádělo validaci pokaždé zbytečně. Kromě běžných uživatelů mohou knihovnu využívat i jiné programy, které si samy vždy zaručí validitu YANG dat. Avšak z toho důvodu, že pád serveru při nevalidním vstupu by byl naprosto neakceptovatelný, je při implementaci nutné počítat i s nevalidním vstupem.

4.2 Návrh rozhraní pro tvorbu konfiguračních dat

Původně byl návrh nového rozhraní tvořen pouze aplikací dat, ovšem při práci s YANG daty modelů jsem si uvědomil, že tvorba YANG dat může pro některé uživatele být problémová. Proto jsem navrhl i sadu rozhraní, která slouží k tvorbě YANG dat nejčastěji konfigurovaných uzlů všech tří modelů.

Návrh rozhraní pro tvorbu konfiguračních dat je založený na tom, že si uživatel udržuje konfiguraci, která zpočátku může být i prázdná. Tuto konfiguraci, tedy strom s YANG daty, následně vloží na vstup dané funkce rozhraní, které k tomuto stromu buď přidá nebo odebere daný uzel. Tedy pro každý uzel, pro který bude existovat toto rozhraní pro tvorbu konfiguračních YANG dat, bude existovat i jeho opak — rozhraní, který daný uzel z konfigurace odebere. Rozhraní pro tvorbu konfiguračních dat jsem takto navrhl, aby jej bylo možné použít s druhým navrženým způsobem aplikace konfiguračních dat (viz sekce 4.1), který zahodí předchozí konfiguraci a nahradí ji novou.

Problém, který s tímto návrhem souvisí a s kterým se bude při implementaci potřeba vypořádat, je určení konkrétních instancí uzlů typu seznam (viz sekce 2.1). Tento problém vznikne, pokud bude potřeba vytvořit nebo smazat uzel, který je potomkem nějakého uzlu typu seznam. Dalším problémem může být například pokus o vytvoření instance seznamu s hodnotou klíče, který už existuje. Bude potřeba se rozhodnout, jestli má dané API přepsat daný uzel, nebo jestli vrátit chybu, anebo zvolit jiné chování.

Dále se bude potřeba zamyslet nad tím, pro které uzly má smysl toto API implementovat. Navrhuji rozhraní pro tvorbu konfiguračních dat implementovat pouze pro nejčastěji využívané uzly, bez kterých by se NETCONF server neobešel. Pro transportní protokol SSH by těmito uzly byly uzly z podstromu *server-identity* (viz diagram 3.4) a hlavně tedy *SSH host key*. U protokolu TLS by to měly být hlavně uzly popisující certifikát serveru. Pro oba protokoly by to pak byly uzly popisující autentizaci klientů. Nicméně je potřeba zvolit tyto uzly pečlivě, aby se předešlo velkému množství funkcí v API.

Kapitola 5

Implementace nového konfiguračního rozhraní knihovny libnetconf2

Tato kapitola se věnuje implementaci nového konfiguračního API knihovny podle návrhu. Ovšem návrh nebyl dokonalý, a proto se tato kapitola zaměří i na problémy, na které jsem při implementaci narazil, a na jejich řešení. Nové konfigurační rozhraní knihovny bylo implementováno, stejně jako knihovna, v programovacím jazyce C. Implementace byla podobně jako návrh rozdělena do dvou částí — tvorba a aplikace konfiguračních dat.

Při implementaci jsem hojně využíval knihovnu *libyang* (viz sekce 2.1), jelikož implementuje práci s YANG schématy a daty. Z důvodu naplnění *libyang* kontextu modelem *ietf-netconf-server* spolu s modely, které importuje, a povolení jen podporovaných YANG vlastností jsem prvně implementoval API, které provede tento proces. Výstupem tohoto API je *libyang* kontext, které je možný použít jako vstup do jiných funkcí nového konfiguračního rozhraní. Seznam modelů, které se nahrají do kontextu, spolu s podporovanými YANG vlastnostmi je následující:

- **ietf-netconf-server** – ssh-listen, tls-listen, ssh-call-home, tls-call-home, central-netconf-server-supported,
- **ietf-x509-cert-to-name** – nemá vlastnosti,
- **ietf-crypto-types** – cleartext-passwords, cleartext-private-keys,
- **ietf-tcp-common** – keepalives-supported,
- **ietf-tcp-server** – tcp-server-keepalives,
- **ietf-tcp-client** – tcp-client-keepalives,
- **ietf-ssh-common** – transport-params,
- **ietf-ssh-server** – local-users-supported, local-user-auth-publickey, local-user-auth-password, local-user-auth-none,
- **iana-ssh-encryption-algs** – nemá vlastnosti,
- **iana-ssh-key-exchange-algs** – nemá vlastnosti,

- **iana-ssh-mac-algs** – nemá vlastnosti,
- **iana-ssh-public-key-algs** – nemá vlastnosti,
- **iana-crypt-hash** – crypt-hash-md5, crypt-hash-sha-256, crypt-hash-sha-512,
- **ietf-keystore** – central-keystore-supported, inline-definitions-supported, asymmetric-keys,
- **ietf-truststore** – central-truststore-supported, inline-definitions-supported, certificates, public-keys,
- **ietf-tls-common** – tls10, tls11, tls12, tls13, hello-params,
- **ietf-tls-server** – server-ident-x509-cert, client-auth-supported, client-auth-x509-cert,
- **iana-tls-cipher-suite-algs** – nemá vlastnosti.

Podporované YANG vlastnosti byly vybrány podle toho, co už v knihovně bylo implementováno před rozšířením o nové API. Dále se tento výběr odvíjel od toho, o jaké funkcionality měli uživatelé knihovny zájem. Vlastnosti, které nebyly podporovány v době implementace této práce, mohou být implementovány jako rozšíření knihovny v budoucnu.

5.1 Implementace aplikace konfiguračních dat

Funkce rozhraní pro aplikaci konfiguračních dat mají následující prototypy:

- `int nc_server_config_setup_diff(const struct lyd_node *data)` a
- `int nc_server_config_setup_data(const struct lyd_node *data)`.

Pseudokód implementace první z nich je možné vidět ve výpisu 5.1. Nicméně obě tyto funkce jsou si velmi podobné. Rozdílem v implementaci zmíněných dvou funkcí rozhraní je práce s atributem operace.

Obě tyto funkce rozhraní mají za cíl aplikovat *ietf-netconf-server*, *ietf-keystore* a *ietf-truststore* data, ale zaměřím se pouze na aplikaci dat prvního zmíněného modelu. Cílem obou funkcí je projít do hloubky všechny uzly stromu s YANG daty. Pro každý uzel se zjistí hodnota jeho atributu operace. Následně se dle výpisu 5.1 zavolá funkce `nakonfiguruj_uzel`, která je de facto jeden velký `if-else` výraz, jehož vyhodnocení závisí pouze na názvu zkoumaného uzlu. Po vyhodnocení tohoto výrazu už dílčí neveřejné funkce zajistí nastavení konkrétních hodnot v interní reprezentaci konfigurace *libnetconf2* serveru.

Popsaná funkce, která sama sebe volá rekurzivně, si musí uchovávat operaci rodičovského uzlu. Navržený způsob aplikace konfiguračních dat, který zahodí veškerou předchozí konfiguraci, jsem implementoval tak, že do rekurze propaguje pouze operaci pro vytvoření[1]. U druhého způsobu se vždy musí zjistit atribut operace daného uzlu a mohou nastat dvě situace. První možností je, že zkoumaný uzel operaci neobsahuje, a tedy zdědí operaci od svého rodiče. Druhou možností je, že uzel operaci obsahuje. Podle zjištěné hodnoty operace se provede konfigurace daného uzlu a hodnota rodičovské operace bude pro další rekurzivní volání nahrazena zjištěnou operací. Pokud má zjištěná operace hodnotu „smazat“, tak už

[1] Dle výpisu 5.1 by tedy tento způsob aplikace konfiguračních započal rekurzi s hodnotou parametru `operace_rodice` rovné „vytvorit“ namísto „neznama_operace“.

nedojde k navštívení potomků daného uzlu a celý podstrom bude z konfigurace serveru smazán.

Jelikož knihovna *libnetconf2* podporuje práci s více vlákny, musel jsem implementovat i jejich synchronizaci v rámci práce s konfigurací. Synchronizaci jsem implementoval tak, že jsem přidal zámek, který se uzamkne buď při aplikaci konfigurace nebo při jejím čtení, které nastává například při autentizaci uživatelů.

```
1  int aplikuj_netconf_server_data(uzel, operace_rodice) {
2    operace_uzlu = ziskej_operaci(uzel);
3    if (uzel nema operaci) {
4      operace_uzlu = operace_rodice;
5    }
6
7    switch (operace_uzlu) {
8    case "nic":
9      break;
10   case "vytvorit":
11   case "nahradit":
12   case "smazat":
13     /* provede konfiguraci uzlu podle jeho nazvu a hodnoty operace */
14     nakonfiguruj_uzel(uzel, operace_uzlu);
15     break;
16   }
17
18   /* smazanim rodice uz byli smazani i jeho potomci */
19   if (operace_uzlu != "smazat") {
20     foreach dite in ziskej_deti(uzel) {
21       /* rekurzivni volani na vsechny potomky uzlu */
22       aplikuj_netconf_server_data(dite, operace_uzlu);
23     }
24   }
25 }
26
27 int nc_server_config_setup_diff(yang_data) {
28   uzamkni(konfiguracni_zamek);
29
30   aplikuj_keystore_data(yang_data, neznama_operace);
31   aplikuj_truststore_data(yang_data, neznama_operace);
32   aplikuj_netconf_server_data(yang_data, neznama_operace);
33
34   odemkni(konfiguracni_zamek);
35 }
```

Výpis 5.1: Pseudokód implementace aplikace konfiguračních YANG dat.

5.2 Problémy s algoritmem pro aplikaci konfiguračních dat

Při implementaci navrženého algoritmu pro aplikaci YANG dat jsem narazil na problémy, na které jsem při návrhu nepřišel. Realizace jejich řešení byla celkem jednoduchá, avšak implementace těchto řešení si prošla několika iteracemi, než se dostala do finální podoby.

První problém je založen na tom, že v modelovacím jazyce YANG nemusí nutně být názvy uzlů unikátní. Například název uzlu *name* se může několikrát nacházet v tom samém podstromu. Jelikož algoritmus, který jsem navrhl a implementoval, je založen na rozlišení uzlů podle jejich názvů, tak problém, který jsem musel vyřešit, byl vypořádání se s multiplicitou názvů uzlů. Implementace řešení tohoto problému spočívá v počítání rodičovských uzlů mezi daným uzlem a nějakým společným rodičem. Tento problém je takto možné řešit, jelikož počet rodičů se dá dopředu určit z popisu modelu, kterému daná data náleží.

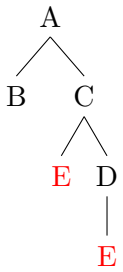


Diagram 5.1: Příklad datového stromu s kolizí názvů uzlů.

V diagramu 5.1 je možné vidět strom, který obsahuje dva uzly s názvem `E`. Funkce obsluhující konfiguraci uzlu `E` musí umět rozlišit, o jaký uzel se jedná, jelikož by tyto uzly mohly mít rozdílný význam. V tomto případě bych si v implementaci jako společný rodičovský uzel určil například `C` a podmínkou pro rozhodnutí, o který uzel `E` se jedná, by byl počet rodičů mezi uzly `E` a `C` — buď jedna, anebo dva.

Toto řešení je funkční, jelikož jsem si v modelech ověřil, že se na jedné úrovni nenachází dva uzly se shodným názvem. Nicméně mělo jednu velkou nevýhodu a to, že v době implementace této práce ještě nebyl *ietf-netconf-server* YANG model standardizován. To znamenalo, že názvy uzlů anebo jejich hierarchie se několikrát změnila. Po standardizaci to ovšem nebude problémem.

Dalším problémem při aplikaci konfiguračních dat, který jsem řešil, bylo dohledání konkrétní instance uzlu typu seznam, jež je popsaný v sekci 2.1. Uzly typu seznam se v YANG datech mohou vyskytovat mnohokrát a jejich podstromy se mohou lišit pouze v hodnotách jejich klíčů. Tudíž je při aplikaci dat daného podstromu potřeba určit, o kterou instanci seznamu se jedná.

Dle diagramu 5.2 uvažujme situaci, kdy je právě obsluhován uzel s názvem `Y`. Z principu algoritmu DFS (*Depth First Search*), který jsem v implementaci využil pro aplikaci YANG dat, už museli být nakonfigurováni všichni předci uzlu `Y`, což v implementaci znamená, že už například byla vytvořena datová struktura pro seznam `koncový-bod` a že už byl uložen jeho klíč. Řešení problému získání konkrétní instance jsem implementoval tak, že procházím všechny rodiče problémového uzlu, dokud nenarazím na daný rodičovský uzel typu seznam a získám hodnotu jeho klíče. Tuto hodnotu pak použiju při prohledávání interních datových struktur pro daný seznam, dokud nenastane shoda klíčů.

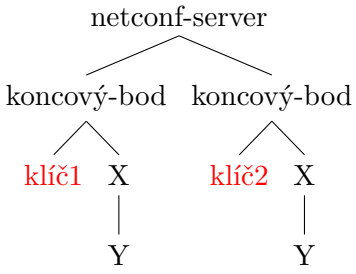


Diagram 5.2: Příklad datového stromu s více instancemi uzlu typu seznam.

5.3 Implementace tvorby konfiguračních dat

Sadu API pro tvorbu konfiguračních dat jsem implementoval tak, že uživatel dodá na vstup rozhraní strom s YANG daty. K tomuto stromu jsou pak přidány uzly dle daného rozhraní a dalších vstupních parametrů. Tento vstupní strom může být i prázdný. V takovém případě bude pomocí rozhraní vytvořen a vrácen zpět uživateli. Dále jsem implementoval i API pro mazání YANG dat ze stromu s daty, který je vstupem těchto rozhraní. Například implementované API pro smazání koncového bodu smaže z YANG dat daný koncový bod spolu se všemi jeho potomky. Naopak při tvorbě uzlu současně dojde i k vytvoření všech jeho rodičovských uzlů.

Při tvorbě YANG dat je problém s instancemi uzlu typu seznam (viz sekce 2.1) řešen tak, že dané rozhraní bere jako parametry klíče všech seznamů, jež jsou rodiči daného vytvářeného (smazávaného) uzlu (uzlů). Pokud instance s daným klíčem existuje, její obsah bude změněn, jinak bude vytvořena. Rozhraní pro mazání uzlů ze stromu s YANG daty vrátí chybu, pokud se nepodaří najít instanci s daným klíčem.

Každá funkce API pro tvorbu dat má oproti svému protějšku, tedy API pro mazání dat, o jeden parametr navíc, kterým je *libyang* kontext (viz sekce 2.1). Ten je při tvorbě YANG dat vyžadován, jelikož knihovna *libyang* potřebuje vědět, jaké typy uzlů jsou vytvářeny. Naopak při mazání uzlů tomu tak není. Dalším důvodem nutnosti tohoto parametru je, že API pro tvorbu dat současně vždy doplní všechny výchozí hodnoty listových uzlů (tedy pouze listů, které nějakou výchozí hodnotu mají).

Výčet uzlů, pro něž existuje API pro tvorbu i mazání konfiguračních YANG dat je následující:

- listy *local-address* a *local-port* (viz diagram 3.3 pro SSH a diagram 3.6 pro TLS)[2],
- seznam *asymmetric-key* (data modelu *ietf-keystore*),
- seznam *certificate* (data modelu *ietf-keystore*),
- seznam *public-key* (data modelu *ietf-truststore*),
- seznam *certificate* (data modelu *ietf-truststore*),
- seznam *host-key* (viz diagram 3.4),
- list *central-keystore-reference* (viz diagram 3.4),

[2]Listy jsou ve výčtu sice uvedeny dva, ale jedná se de facto o API, jehož účelem je vytvořit (smazat) koncový bod, který může být buď SSH, anebo TLS.

- seznam *public-key* (viz diagram 3.5),
- list *password* (viz diagram 3.5),
- list *central-truststore-reference* (viz diagram 3.5),
- kontejner *certificate* (viz diagram 3.7),
- list *central-keystore-reference* (viz diagram 3.7),
- seznam *certificate* v podstromu *ee-certs* (viz diagram 3.8),
- list *central-truststore-reference* v podstromu *ee-certs* (viz diagram 3.8),
- seznam *certificate* v podstromu *ca-certs* (viz diagram 3.8),
- list *central-truststore-reference* v podstromu *ca-certs* (viz diagram 3.8),
- seznam *cert-to-name* (viz diagram 3.6).

Tento výčet je téměř identický s rozhraním tvořícím *Call Home* YANG data. Hlavním rozdílem je, že se zde navíc nachází rozhraní pro tvorbu a mazání instance seznamu *netconf-client*, viz diagram 3.9. Tudíž skoro všechny API pro tvorbu konfiguračních dat týkající se podstromu *Call Home* mají o jeden parametr navíc, kterým je klíč k seznamu *netconf-client* (tedy klíč identifikující NETCONF *Call Home* klienta).

5.4 Vlastní rozšíření ietf-netconf-server YANG modelu

Jazyk YANG umožňuje zasahovat do jiných modelů a měnit tak jejich popis. To je možné za pomoci klíčového slova *augment*, což je detailněji popsáno v sekci 2.1 a v RFC 7950 [2]. Jelikož model *ietf-netconf-server* nepopisuje některé možnosti konfigurace, které se vyskytovaly v původním konfiguračním rozhraní knihovny *libnetconf2* anebo o které uživatelé měli zájem, tak jsem vytvořil vlastní YANG model s názvem *libnetconf2-netconf-server*, jehož účelem je rozšířit popis *ietf-netconf-server* YANG modelu.

Tento nový YANG model momentálně rozšiřuje *ietf-netconf-server* pouze o popis s autentizačním kontextem. Snad nejpodstatnějším rozšířením je definice nového způsobu autentizace u protokolu SSH. Tento popis rozšiřuje seznam *user* (viz diagram 3.5) o tzv. *Keyboard Interactive* autentizační metodu [8]. Toto rozšíření lze vidět na diagramu 5.3.

Dalším rozšířením je přidání možnosti nakonfigurovat si maximální počet neúspěšných pokusů při autentizaci a maximální dobu trvání autentizace u protokolu SSH. Tato rozšíření mají za cíl zvýšit bezpečnost a stabilitu systému. Konfigurace maximálního počtu neúspěšných autentizačních pokusů do jisté míry chrání před tzv. *brute-force* útokem, kde se útočník snaží opakovaně hrubou silou uhodnout zejména heslo. Účelem maximální doby trvání autentizace je prevence před tzv. DoS (*Denial of Service*) útokem, jelikož neaktivní pokusy o autentizaci mohou způsobit zbytečné zatížení a v extrému až výpadek serveru. Toto rozšíření modelu *ietf-netconf-server* je součástí diagramu 5.3.

Další rozšíření se týká definice nových YANG identit [2], které reprezentují algoritmy pro zabezpečení SSH komunikace, viz odstavec o těchto algoritmech v sekci 3.2.1. Tyto nové identity jsem definoval, aby bylo pomocí knihovny *libssh* možné nakonfigurovat všechny algoritmy, které podporuje.

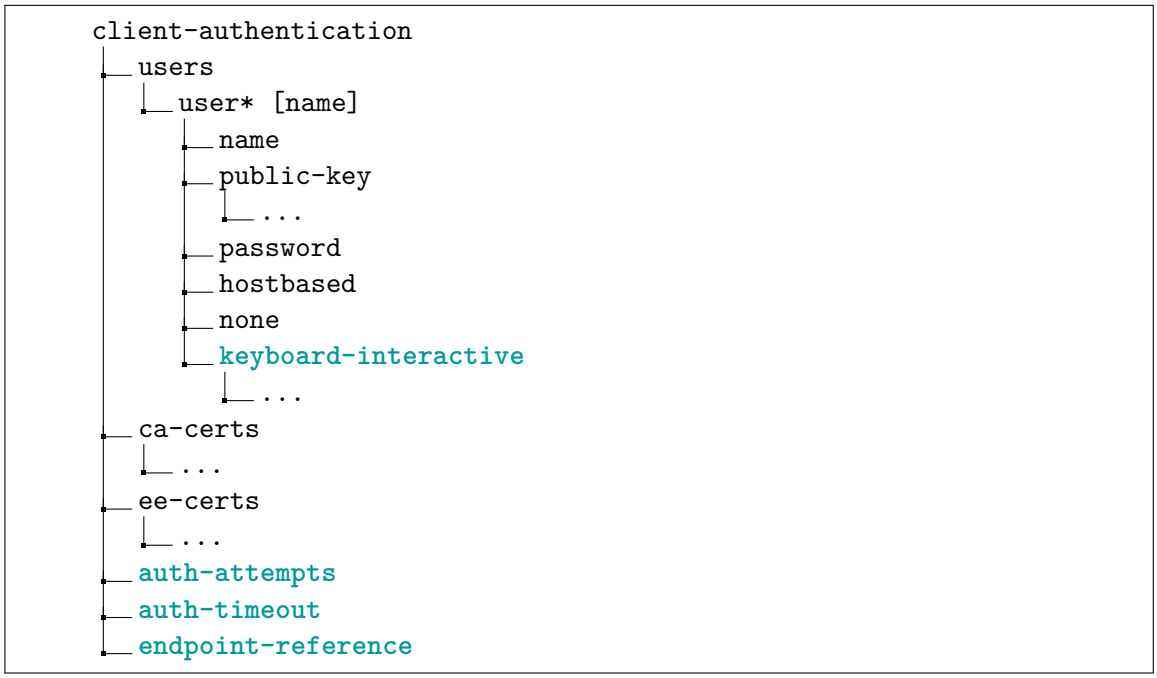


Diagram 5.3: Struktura rozšířeného kontejneru *client-authentication* pro podstrom *ssh*. Uzly rozšiřující původní popis jsou barevně a tučně zvýrazněny.

Následně jsem přidal popis pro seznam zneplatněných certifikátů [4]. Tento seznam slouží k zneplatnění klientských certifikátů. Pokud by se sériové číslo klientem prezentovaného certifikátu vyskytovalo v jednom z těchto nakonfigurovaných seznamů, tak klientovi bude zamítnut přístup. Navrhl a popsal jsem tři způsoby[3] získávání tohoto seznamu, a to buď pomocí odkazu[4], ze kterého bude seznam stažen, nebo pomocí nakonfigurování cesty k lokálnímu souboru, anebo z rozšíření certifikátů s názvem *Distribution Points* [4]. Toto rozšíření modelu je možné vidět na diagramu 5.4.

Předposledním rozšířením bylo přidání dvou identit pro formáty privátních klíčů, aby uživatelé měli větší volnost a mohli konfigurovat privátní klíče v jiných formátech, než v těch, které jsou definované YANG modelem *ietf-crypto-types*. A poslední novinkou je přidání možnosti odkazovat se z jednoho koncového bodu na konfiguraci autentizace jiného koncového bodu. Tyto koncové body musí být naslouchací a používat stejný protokol (tj. buď SSH, nebo TLS). Současně odkazovaný koncový bod se může odkazovat na jiný, avšak tyto reference nesmí tvořit cyklus. Toto rozšíření je možné vidět na diagramu 5.3 pro SSH a na diagramu 5.4 pro TLS. Všechna tato rozšíření platí jak pro naslouchací, tak i pro *Call Home* mód.

[3] Je nutno dodat, že se jedná o výběr jedné ze tří možností, a to za pomoci tzv. *choice* uzlu, viz RFC 7950 [2].

[4] Přesněji řečeno za pomoci URL (*Uniform Resource Locator*).

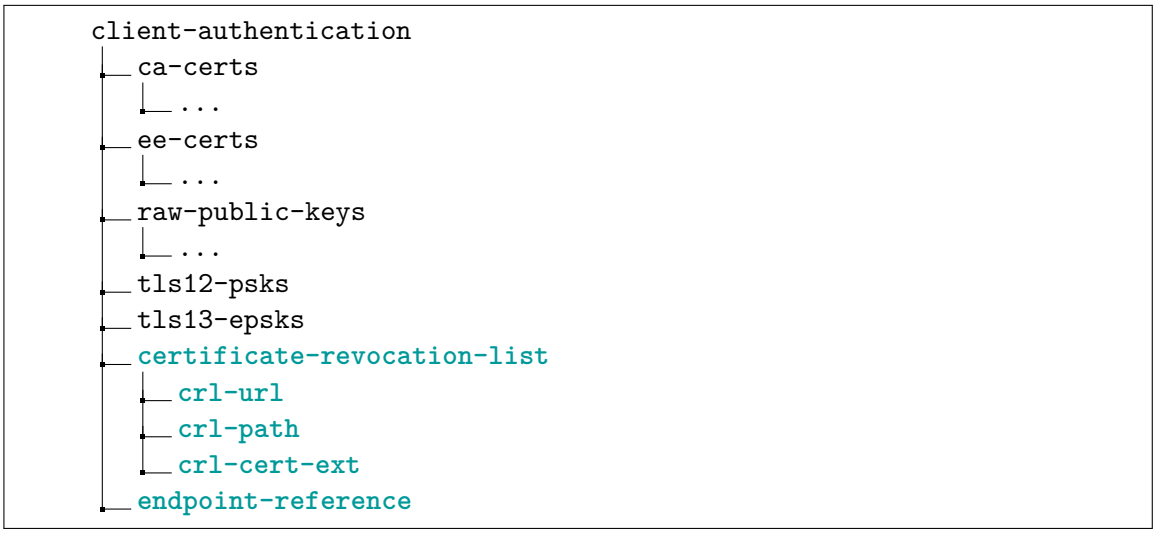


Diagram 5.4: Struktura rozšířeného kontejneru *client-authentication* pro podstrom *tls*. Uzly rozšiřující původní popis jsou barevně a tučně zvýrazněny.

5.5 Přidání podpory pro různé formáty asymetrických párů klíčů

Při práci na implementaci aplikačního rozhraní pro tvorbu konfiguračních YANG dat jsem se zamýšlel nad tím, jak tento proces co nejvíce usnadnit uživatelům. YANG model *ietf-netconf-server* popisuje použití pouze dvou formátů veřejných a dvou formátů privátních klíčů. Navíc první z těchto formátů pro veřejné klíče může být použitý pouze v rámci SSH a druhý pouze v TLS. Zamyslel jsem se nad tím, jestli má smysl takto omezovat i uživatele. Rozhodnutím bylo kompletně abstrahovat formáty klíčů. Proto jsem implementoval API, jehož uživatelským vstupem je cesta k souboru s klíčem. Tento klíč jsem pak, pokud to bylo nutné, převedl do správného formátu a vrátil uživatelovi validní YANG data.

Podmínkou pro úspěšné zpracování klíče je, aby byl uložený ve formátu `PEM` (*Privacy-Enhanced Mail*) [10]. Formát `PEM` reprezentuje data v tisknutelném `ASCII` formátu a obvykle se používá pro ukládání kryptografických dat. Obsah souboru, který obsahuje ať už veřejný nebo privátní klíč zakódovaný v `PEM` formátu, je typicky tvořen hlavičkou a patičkou, které udávají formát klíče. Mezi nimi se nachází data klíče zakódovaná v *Base64* [9], ale mimo jiné se zde mohou nacházet i například komentáře.

Jelikož jsem při implementaci využíval knihovny *OpenSSL* a *libssh*, tak jsem se rozhodl podporovat pouze formáty klíčů, které je možné zpracovat pomocí rozhraní těchto knihoven. Prvně popíšu podporované formáty privátních klíčů.

Podle popisu *ietf-netconf-server* YANG modelu je možné nakonfigurovat pouze privátní klíče ve formátech `PKCS#1` [13] a `SEC1` [5]. Nicméně v současnosti se často používají další dva formáty privátních klíčů, které jsem se rozhodl podporovat. Těmito formáty jsou `OpenSSH` [14] a `PKCS#8` [18]. Přehled těchto formátů, jejich hlaviček a patiček, je možné vidět v tabulce 5.1. Tyto formáty klíčů jsem musel popsat formou identit v jazyce YANG a tento popis se nachází v YANG modelu *libnetconf2-netconf-server* (viz sekce 5.4).

Převod privátního klíče ze souboru do YANG dat jsem implementoval tak, že prvně načtu první řádek souboru, který je uživatelským vstupem, a porovnám ho s hlavičkami v tabulce 5.1. Pakliže najdu shodu, tak se rozhodnu, kterou knihovnu použiju pro načtení

tohoto souboru s klíčem. Všechny jmenované formáty klíčů, až na formát `OpenSSH`, je možné načíst pomocí knihovny *OpenSSL*. S klíčem ve formátu `OpenSSH` si však poradí knihovna *libssh*. Pokud je klíč úspěšně načten, tak pomocí použité knihovny získám nazpět *Base64* zakódovaná data privátního klíče, která použiju jako hodnotu nově vytvořeného listu YANG dat.

Existují dva důvody k tomu, proč jsem se nezbavil hlavičky a patičky a data klíče rovnou nepoužil k vytvoření YANG dat, a to tedy bez použití knihoven ať už *libssh*, nebo *OpenSSL*. Prvním je, že knihovna zkontroluje, jestli se v daném souboru nachází validní privátní klíč. Druhým důvodem je veřejný klíč. Nepovinným parametrem pro tvorbu některých konfiguračních dat je cesta k souboru s veřejným klíčem. Tento parametr je nepovinný, jelikož veřejný klíč je možné obvykle jednoduše získat z privátního klíče. Tuto vlastnost jsem v implementaci hojně využíval, ale potřeboval jsem k tomu datovou strukturu knihovny pro privátní klíč, kterou daná knihovna využívá pro reprezentaci klíče. Tuto datovou strukturu bych nezískal bez použití dané knihovny.

Název formátu	Hlavička	Patička
`PKCS#1` [13]	-----BEGIN RSA PRIVATE KEY-----	-----END RSA PRIVATE KEY-----
`SEC1` [5]	-----BEGIN EC PRIVATE KEY-----	-----END EC PRIVATE KEY-----
`OpenSSH` [14]	-----BEGIN OPENSSH PRIVATE KEY-----	-----END OPENSSH PRIVATE KEY-----
`PKCS#8` [18]	-----BEGIN PRIVATE KEY-----	-----END PRIVATE KEY-----

Tabulka 5.1: Knihovnou *libnetconf2* podporované formáty privátních klíčů, jejich hlavičky a patičky.

S veřejnými klíči to bylo o něco náročnější. Model *ietf-netconf-server* definuje dvě identity pro popis formátů veřejných klíčů, kterými jsou *ssh-public-key-format* [12] a *subject-public-key-info-format* [4]. Dle popisu modelu je možné použít první z nich pouze v rámci protokolu SSH. Jeho data nesou i název použitého formátu (např. *ssh-rsa*). Naopak druhý z nich je možné použít pouze u TLS. Avšak implementoval jsem převod mezi těmito formáty. Tedy uživatel může jednoduše za použití nového rozhraní použít jeden veřejný klíč jak u SSH, tak i u TLS.

Pokud uživatel nespecifikuje cestu k souboru s veřejným klíčem, bude vygenerován z klíče privátního. Implementace generování veřejného klíče ve formátu *subject-public-key-info-format* byla poměrně jednoduchá, jelikož si s tím dokáže poradit knihovna *OpenSSL*. Oproti tomu s tvorbou veřejného klíče ve formátu *ssh-public-key-format* bylo práce více. Podle typu privátního klíče jsem z něj pomocí knihovny *OpenSSL* získal parametry veřejného klíče, tedy například u typu klíče RSA těmito parametry jsou čísla `n` (modul) a `e` (exponent). Tyto parametry spolu s typem veřejného klíče jsem pak podle specifikace standardu[5] vhodně poskládal a zakódoval do Base64. Výsledkem byla data, která vyhovují popisu identity *ssh-public-key-format*. Tato zakódovaná data jsem následně využil pro vytvoření listu v YANG datech a dal je na výstup daného API.

Při práci s asymetrickými páry klíčů jsem narazil na několik chyb v knihovně *libssh*. První dvě z nich souvisely s neuvolňováním dynamicky alokované paměti. Ztráta činila necelých 7kB dat. Jelikož knihovna *libssh* je *open-source* projektem, rozhodl jsem se tyto chyby opravit sám a přispět tak k tomuto projektu. Na další, poněkud závažnější, chybu jsem narazil při implementaci podpory pro privátní i veřejné klíče typu *ED25519* [11]. Kvůli chybě v knihovně *libssh* nebylo možné nastavit tento typ klíče jako *SSH host key* serveru. O této chybě jsem informoval správce projektu *libssh* a po krátké diskuzi se jim povedlo

[5]Například pro RSA je tento formát veřejného klíče popsán v sekci 6.6 RFC 4253 [12].

tuto chybu opravit. Poslední chyba souvisela se stejným typem klíčů, avšak tentokrát se jednalo o generování veřejného klíče z privátního, a to u *ED25519* klíče ve formátu `PKCS#8`, viz tabulka 5.1. Chyba byla úspěšně opravena. K nahlašování těchto problémů jsem využil online repozitář[6] projektu *libssh*, který se nachází na portálu *GitLab*.

5.6 Příspěvek do návrhu modelu ietf-netconf-server

Tvorba RFC (*Request For Comments*) je proces, který začíná návrhem a postupuje skrze diskuze a revize od mnoha lidí, než se stane oficiálním standardem. Celý tento proces je otevřený a transparentní, což umožňuje zapojení širokého spektra lidí a odborníků. Tato otevřenost pomáhá zajistit, že standardy definované v RFC jsou pečlivě zvažované a přijaté komunitou. V současnosti se k diskuzi ohledně návrhu modelu *ietf-netconf-server* a celkově protokolu NETCONF využívá e-mailová konference.

K diskuzi ohledně návrhu modelu jsem se připojil i já spolu se svým konzultantem. Při implementaci konfigurace *SSH host key* jsem narazil na následující. *SSH host key* je privátní klíč, který tvoří identitu SSH serveru. Při vytváření SSH spojení zašle server klientovi svůj veřejný klíč, který klient využije pro tvorbu symetrického klíče (za účelem zabezpečení komunikace). Tento veřejný klíč je obvykle vygenerován z privátního klíče knihovnou implementující SSH. Touto knihovnou byla v mém případě *libssh*. Nicméně část YANG modelu popisující tento *SSH host key* specifikovala, že veřejný klíč a jeho typ je povinný. Tento typ navíc musel být *ssh-public-key-format*, viz sekce 5.5.

Povinnost těchto uzlů přinesla spoustu implementačních problémů, které se týkaly manuální tvorby veřejných klíčů z různých typů privátních klíčů, a to vše z důvodu, abych mohl vytvořit validní YANG data *ietf-netconf-server* modelu. Zahájili jsme tedy diskuzi ohledně popisu těchto veřejných klíčů. Navrhli jsme buď úplné smazání veřejného klíče z popisu *ietf-netconf-server* YANG modelu nebo alespoň změnu popisu typů a hodnot veřejných klíčů tak, aby byly nepovinné.

Na naše návrhy jsme obdrželi odpověď, která tvrdila, že veřejný klíč nemůže vždy být vygenerován z privátního klíče. Privátní klíč nemusí být vždy čitelný z YANG dat z důvodu přístupových práv nebo může být zašifrovaný pomocí jiného klíče. Tyto situace vedou k tomu, že veřejný klíč občas musí být obsažen v YANG datech. Proto první navržené řešení, smazání veřejného klíče z popisu, nebylo reálnou možností.

Po další diskuzi jsme došli k závěru, se kterým jsme byli spokojeni. Co se podstromu s názvem *host-key* (viz diagram 3.4) týče, tak veřejný klíč spolu s jeho typem se staly nepovinnými uzly. Současně se zde vyskytla nová podmínka, která říká, že pokud YANG data budou obsahovat veřejný klíč, tak jeho typem musí být *ssh-public-key-format*. Podstrom *host-key* má svou obdobu i u TLS, kde byly provedeny obdobné změny, ale případný typ veřejného klíče musí být *subject-public-key-info-format*. Tyto změny jsou součástí YANG modelu *ietf-crypto-types* od jeho 28. verze.

Celá tato diskuze proběhla asi osm měsíců před psaním této zprávy a za tu dobu jsem přišel na to, že vygenerovat veřejný klíč v daném formátu z privátního klíče pomocí knihoven *OpenSSL* a *libssh* není tak obtížné. Takže tato diskuze vůbec nemusela vzniknout, nicméně v e-mailové konferenci NETCONF skupiny jsme přišli na kompromis, který je dle mého dobrým příspěvkem do návrhu *ietf-netconf-server* YANG modelu.

Do e-mailové konference protokolu NETCONF jsem psal ještě jednou, když jsem narazil na jiný problém. Jednalo se o chybu v modelu *ietf-ssh-common*, v němž se nachází popis

[6] Dostupné z https://gitlab.com/libssh/libssh-mirror.

pro RPC generující veřejné klíče. Toto RPC obsahuje volbu typu privátního klíče a všiml jsem si, že volba „nezašifrovaný privátní klíč“ je závislá na podporování YANG vlastnosti „šifrované privátní klíče“, což nedávalo smysl. Tyto chyba byla rychle opravena a je součástí modelu *ietf-ssh-common* od jeho 34. verze.

5.7 Další změny v knihovně libnetconf2

Tato sekce je věnována změnám, o kterých bych se v této práci rád zmínil, ale které úplně nesouvisí s implementací nového konfiguračního rozhraní. Většina změn se týká autentizace ze strany serveru, ale některé změny se dotkly strany klientské.

Snad největší změnou z této sekce je celkově autentizace u SSH. Původně knihovna získávala uživatele, kteří se mohou autentizovat, ze systému. Při autentizaci požádal *libnetconf2* server klienta například o heslo, které porovnal se systémovým heslem daného uživatele. Podpora konfigurace popsané modelem *ietf-netconf-server* vyžadovala předělat způsob získávání těchto uživatelů a jejich podporovaných autentizačních metod. Dalším rozdílem je, že v původním stavu knihovny k úspěšné autentizaci stačilo, aby se uživatel autentizoval pomocí jakékoliv jedné z povolených globálních metod. Nyní musí úspěšně splnit všechny jeho nakonfigurované autentizační metody.

Následující změna se týkala klientského mechanismu protokolu SSH pro ověření identity serveru. Tento mechanismus je založený na ukládání záznamů o důvěryhodných hostitelích a slouží k prevenci tzv. „Man-in-the-middle“ útoku. Každý záznam je tvořen dvojicí identifikátoru hostitele (například IP adresa nebo doménové jméno) a odpovídajícího veřejného klíče hostitele. Záznamy se typicky ukládají do souboru s názvem „known_hosts“.

Původně knihovna implementovala vyhodnocení důvěryhodnosti hostitele pouze na bázi zeptání se uživatele[7]. To znamená, že pokud se chtěl klient připojit na server, o němž nebyl nalezen záznam, tak byl uživatel dotázán, jestli si přeje přidat nový záznam a pokračovat, nebo jestli se má spojení ukončit. Nicméně rozhodl jsem se implementovat čtyři další módy kontroly, jejichž účelem bylo eliminovat potřebu nechávat tuto práci na uživateli.

První z nových módů je striktní, což znamená, že pokud se záznam o hostiteli nenajde, tak se spojení ukončí. Další mód je takový, který bez zeptání se uživatele automaticky přidá záznam, ale pouze o novém hostiteli. Pokud by se našel záznam s daným identifikátorem, ale s neshodujícím se veřejným klíčem, tak by se spojení neprovedlo. Ještě volnějším módem je takový, který přidá nový záznam vždy, a tedy i tato kontrola je vždy úspěšná. Poslední mód je takový, který neprovádí žádnou kontrolu, a tedy ani nepřidává nové záznamy. Inspirací pro tyto módy kontroly byl unixový program `ssh`, jehož součástí je i manuálová stránka `ssh_config`, která obsahuje popis těchto módů.

5.8 Závislosti knihovny libnetconf2

Pro sestavení knihovny *libnetconf2* se využívá nástroj *CMake*[8]. *CMake* je nástroj, který umožňuje spravovat projekt a jeho verzi, ověřovat výskyt a verzi závislostí programu, vytvořit tzv. *Makefile*, pomocí nějž dokáže překladač přeložit zdrojový kód do strojového, a mimo jiné i automatizovat testování. Současně vyžadovanou minimální verzí programu *CMake* je verze 3.5.

[7] Pokud pominu možnost, že existovalo rozhraní, pomocí nějž si mohl uživatel toto chování definovat sám.
[8] Dostupné z https://cmake.org/.

Pomocí jazyka *CMake* je možné popsat různé podmínky nutné pro sestavení daného projektu a celkově i způsob tohoto sestavení. Jedna z těchto podmínek se váže na operační systém. Je nutné, aby vycházel z operačního systému UNIX. Mimo *CMake* má *libnetconf2* pouze dvě další závislostí nutné pro sestavení, jimiž jsou překladač jazyka C (např. *gcc*, verze alespoň 4.8.4) a knihovna *libyang*[9], jejíž minimální potřebná verze v době psaní této práce je 2.28.0.

Mezi volitelné, ale silně doporučené, závislosti projektu *libnetconf2* dále spadají knihovny *libssh*[10], *OpenSSL*[11] a *libcurl*[12], bez nichž není možné využít ani SSH, ani TLS spojení. Jejich minimální požadované verze jsou postupně 0.9.5, 3.0.0 a 7.30.0. Knihovna má i další méně významné volitelné závislosti, mezi něž bych zařadil například knihovnu *libpam*, která slouží jako další způsob autentizace u protokolu SSH.

[9] Dostupné z https://github.com/CESNET/libyang.
[10] Dostupné z https://www.libssh.org/.
[11] Dostupné z https://www.openssl.org/.
[12] Dostupné z https://curl.se/.

Kapitola 6

Testování nového konfiguračního rozhraní knihovny

Testování implementace probíhalo za pomoci vlastní testovací sady. Mnou vytvořená testovací sada byla budována postupně, tedy snažil jsem se vždy po implementaci nějaké zásadní části napsat i nový test. K testování v práci využívám dva nástroje, kterými jsou *cmocka*[1] a *CTest*[2]. *CTest* je program, jehož účelem je řídit testování, spouštět jednotlivé testy a pracovat s výpisy. Spouštění jednotlivých testů je myšleno tak, že spouští jednotlivé programy, které vznikly překladem ze zdrojového kódu. Naopak knihovna *cmocka* umožňuje definovat více testů v jednom zdrojovém souboru, vytvořit pro ně specifické prostředí, ve kterém jsou spouštěny, nahrazovat mimo jiné systémová volání vlastními (tj. například donutit je selhat) a dovoluje použít různé podmínky pro kontrolu správného chování testovaného kódu.

Výčet a stručný popis jednotlivých souborů obsahující mnou vytvořené testy je následující:

- test_auth.c – testuje různé způsoby autentizace u SSH,
- test_authkeys.c – testuje autentizaci pomocí napodobených systémových klíčů,
- test_ch.c – testuje *Call Home* spojení,
- test_config_new.c – testuje různé API pro tvorbu konfiguračních dat,
- test_crl.c – testuje použití seznamu zneplatněných certifikátů,
- test_ec.c – testuje užití různých párů klíčů nad eliptickými křivkami o délkách 256, 384 a 521 bitů,
- test_ed25519.c – testuje autentizaci klienta pomocí páru klíčů typu ED25519,
- test_endpt_share_clients.c – testuje sdílení autentizačních způsobů mezi koncovými body,
- test_ks_ts.c – testuje použití dat z datových úložišť *keystore* a *truststore*,
- test_pam.c – testuje autentizaci za využití programu *Linux PAM*,

[1] Dostupné z https://cmocka.org/.

[2] Program *CTest* je součástí projektu *CMake* a jeho dokumentace je dostupná z https://cmake.org/cmake/help/latest/manual/ctest.1.html.

- test_replace.c – testuje náhradu konfigurace serveru před jeho spuštěním,
- test_runtime_changes.c – testuje náhradu konfigurace serveru za běhu,
- test_tls.c – testuje autentizaci a konfiguraci TLS serveru,
- test_two_channels.c – testuje otevření nového NETCONF sezení na již ustanoveném sezení,
- test_unix_socket.c – testuje spojení pomocí unixové schránky.

Všechny zmíněné testy využívají buď nové rozhraní pro tvorbu konfiguračních dat, nebo nové API pro jejich aplikaci, anebo oboje. Zdrojové soubory s testy se z hlediska struktury kódu dost podobají. Jelikož jsem testoval server, tak jsem k tomu potřeboval i klienta. To jsem vyřešil tak, že testy jsou vícevláknové. Nejjednodušší test vytvoří dvě vlákna — první se chová jako server a to druhé jako klient. To ovšem vyžadovalo tato vlákna synchronizovat.

Každý mnou napsaný test obsahuje funkci `main`, která pomocí knihovny *cmocka* definuje jednotlivé testy, které se mají provést. Pro každý jednotlivý test je možné definovat tzv. `setup` a `teardown` funkci. Funkce `setup` se zavolá před provedením daného testu a funkce `teardown` po jeho dokončení. Ve výpisu 6.1 je možné vidět příklad popsané funkce `main`.

Funkce `setup` má v mých testech dva hlavní účely. Prvním je nakonfigurovat server pomocí nového konfiguračního rozhraní. Druhým je vytvořit a inicializovat bariéru podle celkového počtu vláken. Bariéra zastaví klientská vlákna těsně před připojením se na server a pustí je až v ten moment, kdy je server připraven přijímat nová spojení. Funkce `teardown` naopak slouží k uvolnění paměti a dalších zdrojů. Dle výpisu 6.1 by tedy funkce `prvni_test` vytvořila vlákna a počkala na jejich ukončení. Tato vlákna mají za úkol provést daný testovací úkon. Správné chování jednotlivých testů je ověřováno pomocí konstrukcí *assert*[3], tedy například pokud v nějakém testu testuji rozhraní pro vytvoření konfiguračních dat, které má při úspěchu návratovou hodnotu 0, a očekávám úspěch, tak tuto návratovou hodnotu porovnávám s hodnotou 0 pomocí konstrukce *assert*. Pokud by rovnost neplatila, a tedy v kódu rozhraní by se pravděpodobně vyskytovala chyba, tak daný test okamžitě skončí neúspěchem.

```
1  int
2  main(void)
3  {
4      /* definice jednotlivych testu */
5      const struct CMUnitTest testy[] = {
6          cmocka_unit_test_setup_teardown(prvni_test, setup_f, teardown_f),
7          cmocka_unit_test_setup_teardown(druhy_test, setup_f, teardown_f),
8      };
9
10     /* spusteni testu */
11     return cmocka_run_group_tests(testy, NULL, NULL);
12 }
```

Výpis 6.1: Příklad funkce `main`, pomocí níž byla práce testována.

[3]Knihovna *cmocka* definuje několik takovýchto konstrukcí pro různé typy. Například pro celá čísla je to `assert_int_equal(x, y)` a pro ukazatele `assert_non_null(ptr)`.

Test, který bych zde rád vyzdvihl, jsem nazval *test_pam.c*. V tomto testu je ověřována funkcionalita tzv. *Keyboard Interactive* [8] autentizační metody protokolu SSH. Tato autentizační metoda je založena na různých interaktivních výzvách od serveru, na které klient odpovídá. V tomto testu jsou výzvy tvořeny a odpovědi zpracovány systémem *Linux PAM*. Systém *Linux PAM* je založen na knihovnách, pomocí nichž systém poskytuje různým aplikacím autentizaci uživatelů a přidělování práv. Podporu pro tuto funkcionalitu jsem do knihovny *libnetconf2* přidal ještě před tím, než jsem začal pracovat na novém rozhraní.

Pro test jsem tedy vytvořil knihovnu, která obsahuje dvě interaktivní výzvy — napsat uživatelské jméno pozpátku a odpovědět na to, kolik je $1 + 1$. Tato knihovna vznikne přeložením souboru *pam_netconf.c*, který se nachází v repozitáři. Knihovna, podle které se *Linux PAM* řídí, se nastaví voláním funkce `pam_start()`, jež je poskytována rozhraním knihovny *libpam*[4]. Tato funkce přijímá jako parametr název knihovny. Ovšem zde jsem narazil na problém, který spočívá v tom, že knihovna se musí nacházet v systémovém adresáři[5], což by vyžadovalo nainstalovat testovací knihovnu do tohoto adresáře. Řešení tohoto problému spočívá v tom, že ve verzi 1.4 knihovny *libpam* bylo přidáno nové API s názvem `pam_start_confdir()`, které umožňuje specifikovat adresář, ve kterém se knihovna pro autentizaci nachází.

Test *test_pam.c* je tedy přeložen pouze pokud má uživatel nainstalován *Linux PAM* a verze knihovny *libpam* je alespoň 1.4. Jelikož knihovna *libnetconf2* interně volá funkci `pam_start()`, tak bylo potřeba obrátit se na tzv. *linker* (sestavovací program). V souboru *test_pam.c* jsem deklaroval funkci s názvem `__real_pam_start` a definoval funkci `__wrap_pam_start`, viz výpis 6.2. Tyto funkce jsou takto pojmenovány záměrně, jelikož soubor *test_pam.c* je na rozdíl od ostatních zmíněných testů překládán s možnostmi `-Wl,--wrap=pam_start`. Možnost `-Wl` umožňuje předat to, co po ní následuje, spojovacímu programu jako možnost pro něj. Spojovací program tak obdrží možnost `--wrap=pam_start`, která říká, že jakákoliv nedefinovaná reference na `pam_start` se má nahradit symbolem `__wrap_pam_start` a naopak všechny nedefinované reference na `__real_pam_start` se mají nahradit symbolem `pam_start`. To mi de facto umožnilo nahradit interní volání funkce `pam_start()` funkcí `pam_start_confdir()` a otestovat tak jak konfiguraci, tak celkově chování *Keyboard Interactive* autentizační metody pomocí systému *Linux PAM*.

```
1  int
2  __real_pam_start(const char *service_name, const char *user,
3      const struct pam_conv *pam_conversation, pam_handle_t **pamh);
4
5  int
6  __wrap_pam_start(const char *service_name, const char *user,
7      const struct pam_conv *pam_conversation, pam_handle_t **pamh)
8  {
9      return pam_start_confdir(service_name, user, pam_conversation,
           BUILD_DIR "/tests", pamh);
10 }
```

Výpis 6.2: Ukázka kódu umožňující nahrazení volání funkce jinou.

[4]Knihovna *libpam* je součástí většiny operačních systémů založených na systému UNIX, případně je dostupná jakožto balíček.

[5]Na systémech z rodiny UNIX se obvykle jedná o adresář */etc/pam.d*.

Mnou vytvořené testy byly velkou nápomocí při implementaci, avšak současně jejich účelem je odhalit jakékoliv chyby při nových změnách. Ačkoliv všechno, co je v zmíněných testech testováno, úspěšně prochází, tak rozhodně není ověřováno celé nové API, ale pouze jeho významné části. Dle programu *Coverity*[6] se díky novým testům zvýšilo pokrytí testovaného kódu knihovny o více než 11%. Motivací do budoucna je rozšířit testovací sadu o nové testy a dále testovat i případy, které běžně nenastanou. Takovýmto případem může být například selhání alokace paměti. Jednou z možností, jak takovou situaci testovat, je podvrhnout sestavovacímu programu funkci se stejným jménem jako dané systémové volání, což je obdobou toho, co jsem implementoval v testu *test_pam.c*.

[6] Dostupné z https://scan.coverity.com/.

Kapitola 7

Použití nového rozhraní v rámci netopeer2 NETCONF serveru

netopeer2 je projektem, který využívá knihovnu *libnetconf2* pro její implementaci protokolu NETCONF. Dále používá knihovnu *sysrepo* zejména pro její implementaci datových úložišť. Jelikož jsou knihovny *libnetconf2* a *sysrepo* závislé na knihovně *libyang*, tak tato závislost tranzitivně platí i pro *netopeer2*, který ji využívá hlavně pro práci s YANG modely a pro implementaci různých RPC, viz sekce 2.2. Projekt *netopeer2* tedy spojuje všechny tyto knihovny dohromady, čímž dává vzniknout komplexnímu NETCONF serveru.

Prvně je potřeba popsat, jak fungoval *netopeer2* server před mými změnami. *netopeer2* si konfiguraci serveru udržuje ve třech typech datových úložišť, která jsou popsána v sekci 2.2. Z nich nejpodstatnějším typem datového úložiště pro tuto část je tzv. *running*, tedy jinými slovy aktuální konfigurace. Tuto konfiguraci je možné změnit dvěma způsoby. Prvním z nich je klientem zaslané NETCONF RPC žádající o změnu konfigurace. Druhým způsobem je správce datového úložiště, kterým je program *sysrepocfg*, jež je součástí projektu *sysrepo*. Podstatné je, že knihovna *sysrepo* dokáže zaznamenat změny v konfiguraci a informovat o tom *netopeer2* server. Toto informování probíhá tak, že serveru *netopeer2* jsou předána YANG data popisující změny oproti stávající konfiguraci, viz sekce 4.1. Původně *netopeer2* server procházel tato YANG data sám a podle jejich obsahu volal jednotlivé funkce z konfiguračního rozhraní knihovny *libnetconf2*, jež následně nakonfigurovaly server dle změn. Tedy jinými slovy *netopeer2* server de facto transformuje klientské NETCONF požadavky na API volání knihovny *sysrepo* a naopak data získaná od knihovny *sysrepo* transformuje na NETCONF odpovědi.

Po dokončení implementace skoro celého nového API jsem se pustil do jeho integrace v rámci serveru *netopeer2*. To znamenalo rovnou zpracovávat YANG data o změnách konfigurace knihovnou *libnetconf2*, což mi umožnilo smazat tuto funkcionalitu (až několik tisíc řádků kódu) z projektu *netopeer2*. Avšak to nebylo vše. Jelikož *netopeer2* dříve podporoval starší verzi YANG modelu *ietf-netconf-server*, tak bylo potřeba aktualizovat zejména počáteční (tj. *startup*) konfiguraci serveru. Počáteční konfigurace serveru je nastavována při instalaci projektu *netopeer2*, a to za pomoci skriptů, které jsem musel správně upravit. Touto počáteční konfigurací je SSH server v naslouchacím módu s jedním koncovým bodem.

První z těchto skriptů se nazývá *setup.sh*. Jeho hlavním účelem je načíst potřebné YANG modely tak, aby s nimi dokázaly knihovny *sysrepo* a *libyang* správně pracovat. To se provádí pomocí druhého nástroje, který je součástí projektu *sysrepo*, jímž je program *sysrepoctl* — nástroj pro správu a manipulaci s YANG modely. Součástí mé práce bylo přidání YANG

modelu *ietf-netconf-server* a jeho závislostí do projektu *libnetconf2*. Původně však byly součástí projektu *netopeer2* mimo jiné i starší verze těchto modelů, a proto bylo nutné celkově předělat tu část skriptu, která vyhledávala tyto nainstalované YANG modely. Bylo tedy potřeba odstranit YANG model *ietf-netconf-server* a jeho závislosti z projektu *netopeer2* a následně správně načítat tyto modely z adresáře, do kterého byly při instalaci knihovny *libnetconf2* tyto modely nainstalovány.

Druhý skript, který jsem upravoval, se nazývá *merge_hostkey.sh*. Ten se stará o vygenerování privátního klíče, který se využije pro vytvoření YANG dat úložiště *keystore*. Tento klíč je pak v následujícím skriptu odkazován a použit jako SSH *host key*. Posledním skriptem je *merge_config.sh*, jehož účelem je celkově vytvořit konfiguraci s jedním SSH koncovým bodem. V tomto skriptu jsem řešil hlavně autentizaci uživatelů. Pokud se při zpracovávání tohoto skriptu nalezne soubor s autorizovanými veřejnými klíči v očekávaném adresáři uživatele, který tento skript spustil, tak se tyto klíče využijí pro autentizaci. Jinak se autentizace ponechá na knihovně *libnetconf2*, a to za pomoci *Keyboard Interactive* autentizační metody.

Pro zajištění spojení podporuje *netopeer2* kromě protokolů SSH a TLS i použití unixové schránky. Pro úspěšné spojení je nutné, aby klient měl dostatečná práva. Oprávnění schránky je tvořeno hodnotami identifikátoru uživatele (tj. *UID*), identifikátoru skupiny (tj. *GID*) a přístupovými právy. Jelikož ale součástí popisu modelu *ietf-netconf-server* tyto unixové schránky nebyly, tak jsem tento popis přidal do vlastního YANG modelu, viz sekce 5.4.

V rámci projektu *netopeer2* jsou unixové schránky použité v testovací sadě, jelikož skrze ně klient se serverem komunikují. Avšak po nasazení těchto změn jsem dostal zpětnou vazbu od uživatele. Kriticky se vyjádřil k možnosti konfigurovat tuto unixovou schránku pomocí YANG dat. Jeho hlavním argumentem bylo, že tyto nové změny umožňují uživateli s dostatečnými právy vytvořit na systému schránku například s právy správce počítače (tj. *superuser*). Dalším argumentem bylo, že cesta k této schránce je veřejná, jelikož je součástí konfigurace serveru. To se sebou nese jisté riziko z pohledu bezpečnosti, jelikož dochází k úniku informace o systému, na kterém je NETCONF server spuštěn. Zkrátka unixová schránka by neměla být konfigurovatelná pomocí protokolu NETCONF. Uživatelovy argumenty mi přišly pádné, a proto jsem se rozhodl udělat krok zpět a implementovat to původním způsobem — unixová schránka nebude konfigurovatelná pomocí YANG dat, ale bude pro ni existovat speciální API.

Nové rozhraní jsem pomocí programu *netopeer2* prvně testoval za pomoci manuálního upravování jeho počáteční konfigurace. Následně jsem opravil všechny chyby v novém rozhraní tak, aby prošly všechny testy z testovací sady projektu *netopeer2*. Poslední způsob testování spočíval ve změně konfigurace serveru za běhu. Současně jsem spustil dva procesy — *netopeer2-server* (server) a *netopeer2-cli* (klient). Z pozice klienta jsem následně zasílal RPC, které žádaly o změnu konfigurace serveru. Tento způsob testování je popsán obrázkem 7.1. Po přijetí RPC žádající o změnu konfigurace (tj. *edit-config* RPC) se *netopeer2* obrátí na knihovnu *sysrepo*, jejímž účelem je podle dat v RPC zjistit změny oproti stávající konfiguraci a vrátit je zpět *netopeer2* serveru. Tento strom YANG dat popisující změny je posléze předán novému API knihovny *libnetconf2*, jež podle nich provede konfiguraci.

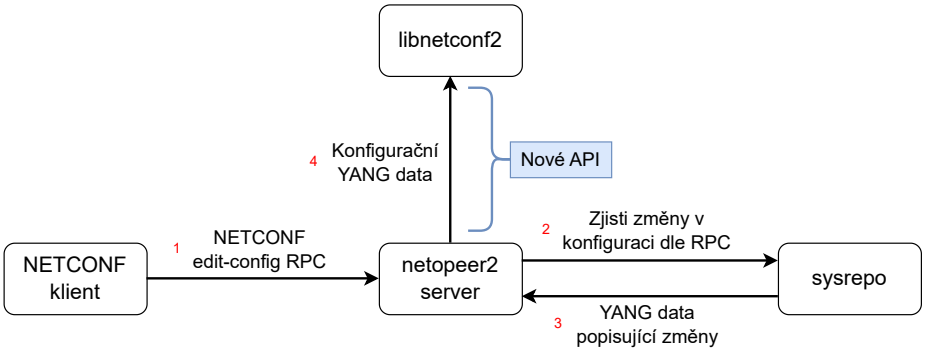
Souhrn nejpodstatnějších změn, které bylo potřeba provést v rámci projektu *netopeer2*, je následující:

- přenechání průchodu změnami v konfiguraci a jejich aplikaci na novém API knihovny *libnetconf2*,
- smazání starší verze modelu *ietf-netconf-server* a jeho závislostí,

- předělání skriptu importujícího YANG modely,
- úprava skriptu vytvářejícího počáteční konfiguraci,
- zajištění správného vyhledání knihovnou *libnetconf2* nainstalovaných YANG modelů,
- úprava testovací sady tak, aby počítala s konfigurací serveru uloženou v datových úložištích,
- aktualizace vzorových konfigurací tak, aby vyhovovaly novější verzi modelu *ietf-netconf-server*.

Užití nového konfiguračního rozhraní knihovny *libnetconf2* v rámci *netopeer2* NETCONF serveru přineslo zejména znatelné snížení řádků, a to nejen kódu, v projektu *netopeer2*. Co se týče zdrojového kódu, tak se počet řádků snížil o `5 324`, jelikož zpracování změn v konfiguraci je ponecháno na knihovně *libnetconf2*. Ve všech souborech projektu pak tento pokles činí `10 744` řádků. To je způsobeno hlavně přesunem některých YANG modelů z projektu *netopeer2* do *libnetconf2*. Z těchto čísel je možné usoudit, že nové API je značně úspornější na použití, co se řádků kódu týče. Nepřímým přínosem, který se na použití nového rozhraní váže, je i redukce potenciálních chyb v implementaci NETCONF serveru.

Verze projektu *netopeer2* integrující nové konfigurační rozhraní knihovny *libnetconf2* je nyní součástí hlavní větve. Od jejího vydání již bylo opraveno několik chyb, z nichž některé měly co do činění s novým rozhraním. Jedna z doposud nejzásadnějších oprav se týkala autentizace pomocí hesla u protokolu SSH, jelikož za jistých jednoduchých podmínek byl vždy při této autentizaci způsoben pád *netopeer2* serveru. Další opravy se hlavně týkaly chyb v zmíněných skriptech.



Obrázek 7.1: Zjednodušené schéma reakce *netopeer2* NETCONF serveru na žádost o změnu konfigurace.

Kapitola 8

Závěr

Cílem této práce bylo seznámit se s knihovnou *libnetconf2*, porovnat její konfigurační rozhraní s popisem modelu *ietf-netconf-server*, z výsledků tohoto porovnání navrhnout nové konfigurační rozhraní knihovny, které umožní nakonfigurovat NETCONF server podle popisu modelu, a následně nové rozhraní podle návrhu implementovat a otestovat. V rámci této práce jsem se seznámil s protokolem NETCONF, modelovacím jazykem YANG, s protokoly SSH a TLS a částečně jsem se ponořil i do asymetrické kryptografie.

Porovnáním původního konfiguračního rozhraní s popisem YANG modelu *ietf-netconf-server* jsem zjistil, že hlavní rozdíl se vyskytuje v konfiguraci klientů, kteří se mohou za pomoci protokolu SSH autentizovat. Nové konfigurační rozhraní jsem navrhl tak, že jeho vstupem jsou YANG data modelu *ietf-netconf-server*. Tato data jsou následně aplikována pomocí průchodu jimi do hloubky a výsledkem je nový stav serveru. Při návrhu jsem si však uvědomil, že vytvořit tato YANG data nemusí být jednoduché, a proto jsem dále navrhl rozhraní, které pomáhá s tvorbou YANG dat použitelných pro konfiguraci.

Nové konfigurační rozhraní se mi úspěšně povedlo implementovat a nyní už je součástí hlavní větve projektu *libnetconf2*. Díky zpětné vazbě uživatelů už bylo opraveno několik chyb. Testování implementace probíhalo dvěma způsoby. První způsob spočíval v postupném budování vlastní testovací sady, pomocí níž jsem testoval nové rozhraní. Druhý způsob byl založen na integraci nového rozhraní do projektu *netopeer2*, kde jsem kromě manuálního testování využíval testovací sadu tohoto projektu.

Práce mi dala široký rozhled, ať už celkově o vývoji aplikací, sítích, síťové komunikaci a síťových protokolech. Dále jsem se dozvěděl i o bezpečnosti a kryptografii. Součástí práce byl i můj příspěvek do *open-source* projektu *libssh* a přispěl jsem i do návrhu modelu *ietf-netconf-server*. Rozsah mé práce přesahuje přes deset tisíc řádků zdrojového kódu, který je nyní součástí *open-source* projektu *libnetconf2*, a tedy musel jsem se současně naučit psát čitelný a udržovatelný kód.

Hlavní motivací do budoucna je uživatelská podpora, která je primárně tvořena odpovídáním na dotazy ohledně přechodu na novou verzi knihovny a opravováním nahlášených chyb. Dalším cílem je rozšířit testovací sadu a tu stávající vylepšit. Účelem je zvýšit množství testovaného kódu, k čemuž bych rád využil i testování případů, které za běžných okolností nenastanou. Dalším možným rozšířením je podobně implementovat i konfiguraci NETCONF klienta, a to podle modelu *ietf-netconf-client*. A v neposlední řadě bych se rád do budoucna dál věnoval implementaci knihovny *libnetconf2* tak, aby byla použitelná na vestavěných zařízeních s omezeným paměťovým prostorem.

Literatura

[1] Badra, M., Luchuk, A. a Schönwälder, J. *Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication* [RFC 7589]. 7589, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, červen 2015 [cit. 2024-03-01]. Dostupné z: https://www.rfc-editor.org/info/rfc7589.

[2] Björklund, M. *The YANG 1.1 Data Modeling Language* [RFC 7950]. 7950, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, srpen 2016 [cit. 2024-02-24]. Dostupné z: https://www.rfc-editor.org/info/rfc7950.

[3] Björklund, M., Schönwälder, J., Shafer, P. A., Watsen, K. a Wilton, R. *Network Management Datastore Architecture (NMDA)* [RFC 8342]. 8342, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, březen 2018 [cit. 2024-03-14]. Dostupné z: https://www.rfc-editor.org/info/rfc8342.

[4] Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S. et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* [RFC 5280]. 5280, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, květen 2008 [cit. 2024-03-21]. Dostupné z: https://www.rfc-editor.org/info/rfc5280.

[5] Brown, D. R. L. a Turner, S. *Elliptic Curve Private Key Structure* [RFC 5915]. 5915, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, červen 2010 [cit. 2024-03-19]. Dostupné z: https://www.rfc-editor.org/info/rfc5915.

[6] Cullen, M. *Using the NETCONF Protocol over Secure Shell (SSH)* [RFC 6242]. 6242, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, červen 2011 [cit. 2024-02-26]. Dostupné z: https://www.rfc-editor.org/info/rfc6242.

[7] Enns, R., Björklund, M., Bierman, A. a Schönwälder, J. *Network Configuration Protocol (NETCONF)* [RFC 6241]. 6241, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, červen 2011 [cit. 2024-02-17]. Dostupné z: https://www.rfc-editor.org/info/rfc6241.

[8] Forssen, M. a Cusack, F. *Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)* [RFC 4256]. 4256, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, leden 2006 [cit. 2024-04-01]. Dostupné z: https://www.rfc-editor.org/info/rfc4256.

[9] Josefsson, S. *The Base16, Base32, and Base64 Data Encodings* [RFC 4648]. 4648, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, říjen 2006 [cit. 2024-04-01]. Dostupné z: https://www.rfc-editor.org/info/rfc4648.

[10] Josefsson, S. a Leonard, S. *Textual Encodings of PKIX, PKCS, and CMS Structures* [RFC 7468]. 7468, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, duben 2015 [cit. 2024-03-17]. Dostupné z: https://www.rfc-editor.org/info/rfc7468.

[11] Josefsson, S. a Liusvaara, I. *Edwards-Curve Digital Signature Algorithm (EdDSA)* [RFC 8032]. 8032, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, leden 2017 [cit. 2024-03-21]. Dostupné z: https://www.rfc-editor.org/info/rfc8032.

[12] Lonvick, C. M. a Ylonen, T. *The Secure Shell (SSH) Transport Layer Protocol* [RFC 4253]. 4253, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, leden 2006 [cit. 2024-03-03]. Dostupné z: https://www.rfc-editor.org/info/rfc4253.

[13] Moriarty, K., Kaliski, B., Jonsson, J. a Rusch, A. *PKCS #1: RSA Cryptography Specifications Version 2.2* [RFC 8017]. 8017, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, listopad 2016 [cit. 2024-03-19]. Dostupné z: https://www.rfc-editor.org/info/rfc8017.

[14] OpenSSH. *PROTOCOL.key* [online]. červenec 2022 [cit. 2024-03-19]. Dostupné z: https://cvsweb.openbsd.org/src/usr.bin/ssh/PROTOCOL.key?annotate=HEAD.

[15] Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3* [RFC 8446]. 8446, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, srpen 2018 [cit. 2024-03-09]. Dostupné z: https://www.rfc-editor.org/info/rfc8446.

[16] Trevino, H. a Chisholm, S. *NETCONF Event Notifications* [RFC 5277]. 5277, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, červenec 2008 [cit. 2024-02-17]. Dostupné z: https://www.rfc-editor.org/info/rfc5277.

[17] Tschofenig, H. a Eronen, P. *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)* [RFC 4279]. 4279, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, prosinec 2005 [cit. 2024-03-09]. Dostupné z: https://www.rfc-editor.org/info/rfc4279.

[18] Turner, S. *Asymmetric Key Packages* [RFC 5958]. 5958, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, srpen 2010 [cit. 2024-03-19]. Dostupné z: https://www.rfc-editor.org/info/rfc5958.

[19] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition) — w3.org* [online]. únor 2013 [cit. 2024-03-24]. Dostupné z: https://www.w3.org/TR/xml/.

[20] Watsen, K. *NETCONF Call Home and RESTCONF Call Home* [RFC 8071]. 8071, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, únor 2017 [cit. 2024-02-24]. Dostupné z: https://www.rfc-editor.org/info/rfc8071.

[21] Watsen, K. *NETCONF Client and Server Models*. Internet-Draft draft-ietf-netconf-netconf-client-server-30, 1. vyd. Wilmington, DE: Internet Engineering Task Force, prosinec 2023 [cit. 2024-02-12]. Work in Progress. Dostupné z: https://datatracker.ietf.org/doc/draft-ietf-netconf-netconf-client-server/30/.

[22] Watsen, K. *A YANG Data Model for a Keystore*. Internet-Draft draft-ietf-netconf-keystore-29, 1. vyd. Wilmington, DE: Internet Engineering Task Force, prosinec 2023 [cit. 2024-03-04]. Work in Progress. Dostupné z: https://datatracker.ietf.org/doc/draft-ietf-netconf-keystore/29/.

[23] Watsen, K. *A YANG Data Model for a Truststore*. Internet-Draft draft-ietf-netconf-trust-anchors-22, 1. vyd. Wilmington, DE: Internet Engineering Task Force, prosinec 2023 [cit. 2024-03-04]. Work in Progress. Dostupné z: https://datatracker.ietf.org/doc/draft-ietf-netconf-trust-anchors/22/.

[24] Wouters, P., Tschofenig, H., Gilmore, J. I., Weiler, S. a Kivinen, T. *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)* [RFC 7250]. 7250, 1. vyd. Wilmington, DE: Internet Enginnering Task Force, červen 2014 [cit. 2024-03-09]. Dostupné z: https://www.rfc-editor.org/info/rfc7250.

Příloha A

Použití a rozšíření programu yanglint

Tato příloha je věnována programu *yanglint*, který je součástí projektu *libyang*. Primární účel programu *yanglint* je validovat YANG data a modely, avšak program poskytuje i interaktivní mód. V tomto módu je možné procházet různé YANG modely a vypisovat jejich uzly nebo jejich hierarchickou strukturu za pomocí textového uživatelského rozhraní.

V mé práci jsem program *yanglint* využíval jak při analýze modelu *ietf-netconf-server*, tak při jeho porovnání s konfiguračním rozhraním knihovny *libnetconf2*, a hlavně i při implementaci nového API. Při seznamování se s YANG modelem jsem využíval informační výpisy, jejichž účelem je vytisknout popis všech uzlů daného podstromu. Příklad takového výpisu je možné vidět na výpisu A.1. Program *yanglint* jsem takto využíval, jelikož model *ietf-netconf-server* importuje několik dalších modelů, a tedy i jejich definic. Tyto definice bych musel hledat v těchto jednotlivých modelech, kdežto díky programu *yanglint* jsou všechny na jednom místě.

Program *yanglint* dále poskytuje stromový mód, který slouží hlavně k vyobrazení hierarchické struktury daného YANG modelu. Tento mód jsem využíval hlavně při implementaci a jeho příklad je možné vidět na výpisu A.2. Tento mód tiskne typy listových uzlů, klíče uzlů typu seznam a dále YANG vlastnosti, na jejichž podporování je daný podstrom závislý. Dále je na výpisu A.2 možné vidět příkazy, které jsem pro zobrazení výpisu použil. Poslední příkaz[1] umožňuje specifikovat cestu ke kontextovému uzlu, od kterého se bude tisknout. Avšak původně nebylo podporováno našeptávání této cesty, a tedy uživatel musel sám přesně zadat cestu k cílovému uzlu, což nebylo moc uživatelsky přívětivé, jelikož tato cesta může být dost dlouhá.

Rozhodl jsem se tedy přispět k programu *yanglint* a implementoval jsem našeptávání a doplňování cesty k uzlům jak pro informační, tak pro stromový mód. Implementace tohoto rozšíření je založena na zpracování aktuální cesty a následném zjištění všech možných následovníků. Uzly, které mohou následovat, se uživateli zobrazí při stisknutí klávesy tabulátor. Pokud je možný následovník pouze jeden, bude do cesty automaticky doplněn.

[1] Jedná se o příkaz `print -f tree -P /ietf-netconf-server:netconf-server/call-home/netconf-client/reconnect-strategy`. Nápovědu k příkazům programu *yanglint* je možné v interaktivním módu zobrazit pomocí příkazu `help`.

```
> print -f info -P /ietf-netconf-server:netconf-server/call-home/
  netconf-client/reconnect-strategy
container reconnect-strategy {
  config true;
  status current;
  description
  "The reconnection strategy directs how a NETCONF server
  reconnects to a NETCONF client, after discovering its
  connection to the client has dropped, even if due to a
  reboot. The NETCONF server starts with the specified
  endpoint and tries to connect to it max-attempts times
  before trying the next endpoint in the list (round
  robin).";
  leaf start-with {
    type enumeration {
      enum "first-listed" {
        value 0;
        description
        "Indicates that reconnections should start with
        the first endpoint listed.";
       }
      enum "last-connected" {
        value 1;
        description
        "Indicates that reconnections should start with
        the endpoint last connected to. If no previous
        connection has ever been established, then the
        first endpoint configured is used. NETCONF
        servers SHOULD be able to remember the last
        endpoint connected to across reboots.";
      }
      enum "random-selection" {
        value 2;
        description
        "Indicates that reconnections should start with
        a random endpoint.";
      }
    }
    default "first-listed";
    config true;
    status current;
    description
    "Specifies which of the NETCONF client's endpoints
    the NETCONF server should start with when trying
    to connect to the NETCONF client.";
  }
  ...
```

Výpis A.1: Výpis podstromu YANG modelu *ietf-netconf-server* v informačním módu programu *yanglint*.

```
> clear -ii
> add ietf-netconf-server@2023-12-28.yang
> print -f tree -P /ietf-netconf-server:netconf-server/call-home/
  netconf-client/reconnect-strategy

module: ietf-netconf-server
+--rw netconf-server {central-netconf-server-supported}?
  +--rw call-home! {ssh-call-home or tls-call-home}?
    +--rw netconf-client* [name]
      +--rw reconnect-strategy
        +--rw start-with? enumeration
        +--rw max-wait? uint16
        +--rw max-attempts? uint8
```

Výpis A.2: Výpis podstromu YANG modelu *ietf-netconf-server* v stromovém módu programu *yanglint*.

Příloha B

Obsah paměťového média

Adresář `libnetconf2`

Obsahem adresáře `libnetconf2` je verze 3.0.8 projektu *libnetconf2* stažená z jeho online repozitáře[1]. V podadresáři s názvem `src` se nachází zdrojové soubory knihovny. V podadresáři `tests` se nachází testovací sada. Další podrobnosti k adresářové struktuře, návod k překladu a instalaci knihovny jsou součástí souboru `README`.

Adresář `text`

Adresář `text` obsahuje zdrojový tvar písemné zprávy bakalářské práce včetně všech náležitostí nutných pro zpětné vysázení práce. Pro sazbu bakalářské práce byl použit systém LaTeX a poskytovaná šablona.

Soubor `bp-xjanot04.pdf`

Soubor `bp-xjanot04.pdf` obsahuje vysázený text bakalářské práce.

[1] https://github.com/CESNET/libnetconf2