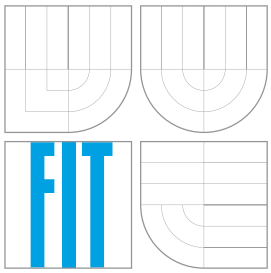
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNTÉZA A VERIFIKACE SÍŤOVÉ KONFIGURACE

NETWORK CONFIGURATION SYNTHESIS AND VERIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. ALŽBETA ČERNEKOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. ONDŘEJ RYŠAVÝ, Ph.D.

BRNO 2016

Zadání diplomové práce/19003/2015/xcerne00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Černeková Alžbeta, Bc.**
Obor: Počítačové sítě a komunikace
Téma: **Syntéza a verifikace síťové konfigurace**
Network Configuration Synthesis and Verification
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s problematikou generování síťové konfigurace. Seznamte se s jazyky pro specifikaci síťových politik, nástroji vhodných pro syntézu konfigurace a modely pro reprezentaci síťové konfigurace.
2. Navrhněte vhodný síťový model pro reprezentaci síťové konfigurace s možností generování konfigurace pro vybranou platformu.
3. Navrhněte jazyk pro definici síťových politik, který by zahrnoval možnost specifikace požadavků a omezení zahrnující směrování a řízení přístupu.
4. Navrhněte a realizujte jako prototyp metodu pro syntézu síťové konfigurace.
5. Na vhodných případových studiích demonstrujte použitelnost navrženého experimentálního řešení.
6. Vyhodnoťte navržené řešení a naznačte možná vylepšení a další vývoj.

Literatura:

- Narain, S., Levin, G., & Malik, S.: Declarative Infrastructure Configuration Synthesis and Debugging. *Journal of Network and Systems*, 1-26, 2008.
- Stone, G. N.: *A Path-based Network Policy Language*. Naval Postgraduate School, 2000.

Při obhajobě semestrální části projektu je požadováno:

- Bez požadavků.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Ryšavý Ondřej, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Predmetom diplomovej práce je problematika konfigurovania siet'ových zariadení. Rieši otázku ako zjednodušit' určité časti konfigurácie a zároveň eliminovat' časté chyby. Predstavuje riešenie nastavovania IP adries na rozhraniach siet'ových zariadení. Ďalej prezentuje generovanie konfigurácie pre určité scenáre dynamického smerovania a jej aplikovanie a metódy na jej verifikáciu na siet'ových zariadeniach. Demonštruje tiež vygenerovanie konfigurácie pre nastavenie VPN tunelu medzi dvomi zariadeniami. V závere obsahuje návrhy d'alších možných rozšírení či vylepšení.

Abstract

The subject of this master's thesis is to address the topic of network devices configuring. It resolves the problems related to simplifying certain parts of configuration while eliminating frequent errors or issues. It introduces a solution to setting of IP addresses on router's interfaces. Furthermore it presents the generation of configuration for particular dynamic routing scenarios and its application and methods for its verification on network devices. It also demonstrates the preparation of configuration of VPN tunnel between two devices. The conclusion of this thesis contains possibilities for further extensions or improvements.

Klíčová slova

siet', konfigurácia, smerovač, generovanie konfigurácie, syntéza, verifikácia

Keywords

network, configuration, router, generating of configuration, synthesis, verification

Citace

ČERNEKOVÁ, Alžbeta. *Syntéza a verifikace síťové konfigurace*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ryšavý Ondřej.

Syntéza a verifikace síťové konfigurace

Prohlášení

Prehlasujem, že som túto diplomovú prácu vypracovala samostatne pod vedením pána doc. Ing. Ondřeja Ryšavého. Uviedla som všetky literárne prameme a publikácie, z ktorých som čerpala.

.....
Alžbeta Černeková
25. května 2016

Poděkování

Chcela by som poďakovať svojmu vedúcemu doc. Ing. Ondřejovi Ryšavému, Ph.D. za pomoc a trpezlivosť pri riešení práce a pri nastavovaní prostredí.

Obsah

Kapitola 1

Úvod

Počítačová sieť je dôležitou súčasťou technického vybavenia organizácie. V súčasnosti veľa organizácií vymienia svoje hardvérové vybavenie za systémy, na ktoré sa pripája. Tento trend prechodu na „cloudové“ riešenia sa však nedá aplikovať na sieťové zariadenia. Skôr naopak, keďže veľa dôležitých systémov je vzdialených, stáva sa stabilná sieť ešte podstatnejšou. Mnoho počítačov dnes plní len funkciu terminálu, no stále je potrebné pripojiť ich a zabezpečiť im prístup na internet.

Či už prístup na internet alebo pripájanie počítačov vyžaduje sieťové zariadenia. Ich počet sa líši v závislosti od veľkosti organizácie, no nie je nezvyčajné hovoriť o desiatkach smerovačov alebo prepínačov, príp. hardvérových firewalloch a pod. Všetky tieto zariadenia vyžadujú konfiguráciu. Nastaviť ju správne býva zvyčajne úlohou sieťových administrátorov. Nastavovanie veľkého množstva zariadení je náchylné na ľudskú chybu, takže môže nastať situácia, kedy sa nechtiac zmení správanie celej siete. Môže ísť o zmeny parametrov v smerovacom protokole alebo o nesprávnu adresu na smerovači, ktorým sa celá organizácia pripája k internetu. Požadované zmeny často vyžadujú úpravu konfigurácie na viacerých zariadeniach, na ktoré sa pripájajú postupne a manuálne zadávajú príkazy.

Aby bolo možné predísť podobným situáciam, jednou z možností je používať softvérové riešenie umožňujúce automatickú konfiguráciu. Môže to byť generovanie konfigurácie nových zariadení, pričom pre všetky nastaví správne počiatočné hodnoty, alebo aplikácia zložitejších konfigurácií na existujúcu sieť. S ohľadom na bezpečnosť firemných dát, ktorá sa často považuje za prioritu, je často nutné nastaviť šifrované spojenie medzi pobočkami. Takáto konfigurácia je komplexná a náchylná chybu. Pokiaľ ju však vygeneruje program na základe vstupných dát, toto riziko sa zníži.

Predmetom tejto práce je riešenie, ktoré nedovolí nahrať konfiguráciu, kde sú vstupné hodnoty v konflikte. Ide o program, ktorý vygeneruje sieťovú konfiguráciu, nastaví ju na jednotlivých zariadeniach a potom zobrazí nastavené informácie, aby ich bolo možné overiť. Vďaka tomu môže uľahčiť prácu so sieťovými zariadeniami.

Kapitola 2

Jazyky na špecifikáciu sieťových politík

Politika (policy) [19] je formálne definovaná ako zoskupenie *pravidiel politiky* (policy rules). Každé pravidlo politiky je tvorené množinou podmienok a zodpovedajúcou množinou akcií. Podmienky definujú, kedy sa pravidlo politiky aplikuje. Keď sa pravidlo politiky aktivuje, môže sa vykonať jedna alebo viacero akcií zahrnutých v tomto pravidle. Tieto akcie sú asociované buď so splnením, alebo s nesplnením množiny podmienok špecifikovaných pravidlom politiky.

Politika [19] je tiež vzťah medzi atribútmi objektov, ktoré sú spravované politikovou aplikáciou, ktorá riadi a ovláda jeden alebo viac aspektov *bodov uplatňovania politík* (policy enforcement point – PEP). Tieto PEP, teda miesta, kde sa uplaňujú politiky, poskytujú služby regulované jednou alebo viacerými politikami. Kľúčovou vlastnosťou tejto definície je schopnosť oddeliť špecifikáciu množiny služieb, ktorá je nezávislá na platforme, od jej implementácie, ktorá sa líši v závislosti od dodávateľa. Politika vytvára spojenie medzi vysokoúrovňovou špecifikáciou požadovaných služieb a nízkoúrovňovou konfiguráciou sieťových zariadení, ktorá má tieto služby zabezpečiť [20].

Podľa definície v [19], politika môže obsahovať politiky. Tento hierarchický prístup umožňuje vytváranie komplexnejších politík z jednoduchších. Takisto povoľuje znovupoužitie stavebných blokov ako sú pravidlá, podmienky či akcie.

Politiky musia spĺňať viaceré dôležité požiadavky [19]:

- použiteľnosť pre veľké distribuované systémy rovnako ako pre menšie riešenia,
- dostupnosť pre entity, ktoré potrebujú prístup,
- bezpečnosť,
- jednoduchá detekcia konfliktov.

2.1 Termy

V roku 1989 predložil D. Clark RFC 1102 [6], v ktorom rozoberá možnosti voľby cesty tak, aby užívatelia (ľudia, služby) mali zabezpečený prístup k sieťovým prostriedkom podľa tried, do ktorých patria. V Clarkovom RFC 1102 je internet reprezentovaný ako viac-menej autonómne *administratívne regióny*. Administratívne regióny majú unikátne identifikátory a sú navzájom prepojené. Aby bola možná komunikácia, musí existovať cesta zo zdrojového

AR do cieľového AR. Clark túto cestu, prechádzajúcu cez viacero AR, nazýva *politikami riadená cesta* (policy route). Na určitej úrovni abstrakcie ide o postupnosť AR. Pred tým, než sa tieto politikami riadené cesty môžu začať používať, je potrebné zaviesť *politikové brány* (policy gateways). Politikové brány overujú platnosť politík a to nasledovne: pozrú sa na zdrojový a cieľový AR a tiež na priamych susedov overovaného AR, ktorí sa nazývajú vstupný a výstupný AR.

2.1.1 Podmienky

Aby mohli termy, ako Clark označuje pravidlá upravujúce sieťovú prevádzku, popisovať reálne politiky, je nutné zahrnúť do zápisu podmienky. Tie určujú, za akých okolností je term platný. Môžu obsahovať typ služby či meniť platnosť termu podľa hodín počas dňa. Základným predpokladom pre funkčnosť politík je ich pochopenie koncovým bodom, ktorý potom dokáže generovať politikou riadenú cestu, ktorá danú podmienku spĺňa. Pri definovaní podmienok sa AR považujú za stabilné entity, ktorých stav sa nemení veľmi často. Umožňuje to dosiahnuť globálnu konzistenciu podmienok.

2.1.2 Príklady termu

V [6] je uvedená notácia, ktorá bola použitá na popis príkladov

$$((\texttt{Hs}, \texttt{ARs}, \texttt{ARent}), (\texttt{Hd}, \texttt{Ad}, \texttt{ARexit}), \texttt{UCI}, \texttt{Cg})$$

kde

Hs, Hd – adresa zdrojového užívateľa, resp. cieľového užívateľa,

ARs, ARd – zdrojový, resp. cieľový AR,

ARent, ARexit – vstupný, resp. výstupný AR,

UCI – *ID užívateľskej triedy* (User Class ID),

Cg – globálna podmienka.

Vo viacerých prípadoch sa používa * ako „žolík“ (wild card), umožňuje teda špecifikovať viacero užívateľov alebo AR, ktoré spĺňajú podmienku. Znak „-“ znamená, že vstupný AR je rovnaký ako zdrojový AR, resp. výstupný AR je rovnaký ako cieľový.
Term

$$((*, *, *), (*, *, *))$$

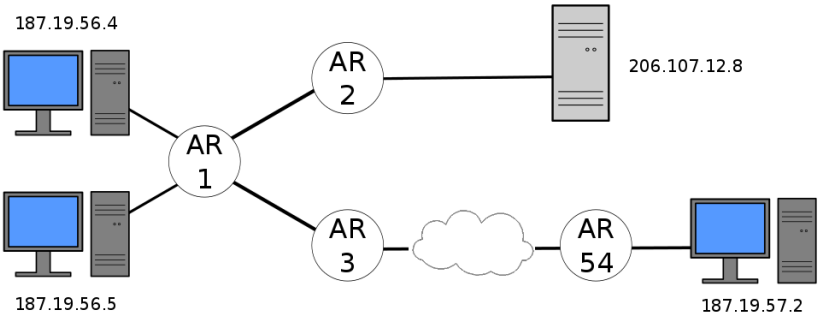
popisuje sieť bez akýchkoľvek obmedzení. V sieti na obr. 2.1 by toto pravidlo povolilo komunikáciu medzi všetkými AR a takisto medzi všetkými užívateľmi. Pravidlo

$$((*, \texttt{1}, -), (\texttt{206.107.12.8}, \texttt{2}, -))$$

dovolí užívateľom pripojeným v AR 1 pristupovať na server s IP adresou 206.107.12.8 v AR 2 a pravidlo

$$((*, \texttt{54}, -), (*, \texttt{3}, *))$$

povolí sieťovú komunikáciu z AR 54, ktorá prechádza cez AR 3.



Obrázek 2.1: Príklad siet'ovej topológie

2.2 PFDL

Policy Framework Definition Language (PFDL, framework pre jazyk na definíciu politík) [20] je založený na tom, že heterogénne zariadenia potrebujú obecnú definíciu politík. Ďalej musí byt' politika reprezentovateľná, spravovateľná jednoznačným spôsobom a schopná spolupracovat', to všetko v rámci jednej administratívnej siet'ovej domény. Schopnost' spolupracovat' znamená v tomto kontexte poskytovanie špecifikácie nezávislej od dodávateľa či zariadenia. Cieľom tohoto jazyku je prekladanie obchodnej špecifikácie do nezávislej intermediárnej formy.

2.2.1 Štruktúra politík v PFDL

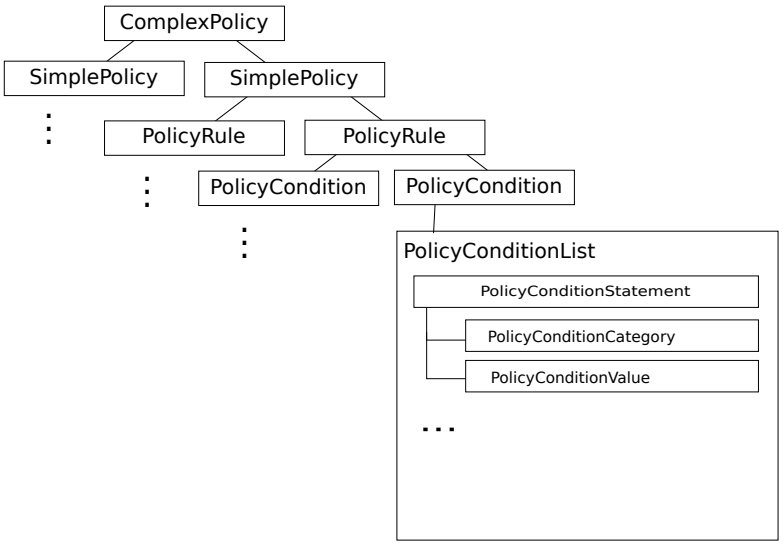
Okrem požiadaviek uvedených v 2.2, PFDL musí byt' škálovateľný a znovupoužiteľný. To vyžaduje všeobecnú definíciu štruktúry politík (obr. 2.2) [20].

Trieda ComplexPolicy

Trieda *ComplexPolicy* [20] (komplexná politika) môže obsahovat' jednu alebo viacero SimplePolicy. Toto umožňuje znovupoužiteľnost' na úrovní politík. [20] uvádza ako príklad využitia situáciu, kedy sa užívateľ prihlási do siete. V tom okamihu sa musia vykonat' viaceré rôzne politiky. ComplexPolicy sa modeluje ako zoskupenie SimplePolicies.

Trieda SimplePolicy

Trieda *SimplePolicy* [20] (jednoduchá politika) obsahuje jedno alebo viacero PolicyRule, pravidiel politiky. Každá SimplePolicy, teda trieda PolicyRule, obsahuje množinu *PolicyConditions* (podmienky politík) a množinu *PolicyActions* (akcie). Dôvodom je, že každé *PolicyRule* je doménovo špecifické. Existujú rôzne PolicyRules, ktoré definujú určité mechanizmy, napr. nastavenia bezpečnosti. Každý z nich má svoju vlastnú definíciu, ktorá je špecifická pre jeho doménu znalostí a ktorá požaduje množinu doménovo špecifických podmienok a akcií. Avšak na abstraktnejšej úrovni určité politiky potrebujú na poskytnutie požadovanej služby viacero agregovaných pravidiel politiky. Na PFDL boli kladené d'alšie požiadavky, a teda možnost' zoskupit' rôzne politiky do jednej konzistentnej, ktorú je možné zasociovat' s užívateľom, a tiež koncept poradia a priority pravidiel politiky. [20]



Obrázek 2.2: Štruktúra politík v PFDL

Trieda PolicyRule

Táto trieda [20] (pravidlo politiky) vyjadruje množinu podmienok politiky, ktoré, keď sú splnené, spúšťajú množinu akcií, ktoré sa majú vykonať. Množina podmienok je použitá ako časť zovšeobecnenej šablóny v rámci každej domény znalostí. Tj. zabezpečenie či DHCP [1] majú vo všeobecnosti svoje vlastné špecifické šablóny, ktoré definujú konkrétne množiny podmienok a k nim prislúchajúcich akcií, ktoré sa musia vykonať, keď sú podmienky splnené.

Hierarchia triedy PolicyCondition

Podmienka politiky [20] je definovaná ako testovanie jedného alebo viacerých hľadísk užívateľov siete a/alebo sieťovej prevádzky za účelom určenia, či sa môže zachovať súčasný stav politík alebo je možné dosiahnuť nový stav. PolicyRule je tvorené jednou alebo viacerými podmienkami. Každú podmienku reprezentuje *PolicyConditionList* (zoznam podmienok politík). PolicyConditionList zoskupuje jeden alebo viacero *PolicyConditionStatement* (príkazy podmienok politík) a PolicyConditionStatement pozostáva z dvoch častí: *PolicyConditionCategory* (kategória podmienky politiky) a *PolicyConditionValue* (hodnota podmienky politiky). PolicyConditionCategories sú preddefinované prvky, ktoré sú špecifické pre určitú doménu znalostí. Utvárajú štruktúru časti šablóny pravidla politiky tvorenej podmienkou politiky. PolicyConditionValues predstavujú hodnoty, ktoré môže PolicyConditionCategory nadobudnúť. Aby bolo možné uplatňovať kontrolu platnosti konkrétnej hodnoty, existujú preddefinované limity a ďalšie parametre. PolicyConditionValues sú prostriedky na reprezentovanie hodnôt špecifických pre užívateľov alebo aplikácie.

[1] Dynamic Host Configuration Protocol – protokol na dynamické odovzdávanie informácií o konfigurácií klientom `https://www.ietf.org/rfc/rfc2131.txt`

Hierarchia triedy PolicyAction

Trieda *PolicyAction* [20] (trieda akcií politiky) je zoznamom jedného alebo viacerých PolicyConditionStatement. Tieto sa môžu vykonávať buď v ľubovoľnom poradí alebo v predpísanom poradí, s využitím špeciálne definovaných atribútov. Akcia politiky [20] je definovaná ako zmena konfigurácie jedného alebo viacerých sieťových prvkov s cieľom dosiahnuť požadovaný stav politík. Jedna alebo viacero akcií tvoria pravidlo politiky. Každú akciu tvorí *PolicyActionList* (zoznam akcií politiky). Ak týchto zoznamov existuje viac, zvyčajne sa vykonávajú v ľubovoľnom poradí. PolicyActionList zoskupuje jeden alebo viaceré *PolicyActionStatement* (príkaz akcie politiky). Podobne ako príkaz podmienky politiky, aj PolicyActionStatement pozostáva z *PolicyActionCategory* (kategória akcie politiky) a *PolicyActionValue* (hodnota akcie politiky).

Detekcia konfliktov

Detekcia konfliktov je kľúčová pri dizajnovaní škáľovateľného a použiteľného frameworku. [20] rozdeľuje konflikty na dva typy: tie, ktoré môžu vzniknúť v rámci politík a tie, ktoré spôsobujú konflikty medzi akciami, ktoré majú prebehnúť na sieti. Označujú sa ako intra-politikové a inter-politikové.

Intra-politikové konflikty sa vyskytnú, keď podmienky dvoch alebo viacerých politík môžu byť zároveň splnené, ale akcie minimálne jednej z politík nemôžu byť vykonané zároveň. Z toho vyplýva niekoľko vecí: jedno alebo viacero pravidiel politík je splnených rovnakou požiadavkou, každá podmienka každého konfliktného pravidla je splnená tou istou požiadavkou, jedna alebo viacero akcií špecifikovaných jednou politikou je v konflikte s jednou alebo viacerými akciami špecifikovanými inou politikou.[20] Intra-politikové konflikty môžu byť vyriešené rôznymi spôsobmi. Najjednoduchším je zmeniť podmienku alebo akciu jednej z konfliktných politík tak, aby už nebola v konflikte s ostatnými podmienkami. Ak však politiky musia zostať neodstrániteľne konfliktné, konflikty je možné odstrániť aplikovaním prvej politiky, ktorá vyhovuje, alebo nájdením všetkých pravidiel, ktoré vyhovujú a použitím pravidla s najvyššou prioritou, alebo použitím ďalších metadát na určenie pravidla, ktoré sa má použiť'.[20]

Inter-politikové konflikty [20] sú tie, ktoré vzniknú pri aplikovaní dvoch alebo viacerých politík, pričom ale výsledkom sú konfliktné konfiguračné príkazy alebo mechanizmy, určené pre jedno alebo viacero zariadení. Politiky ale v tomto prípade nie sú v konflikte, ten vzniká až pri ich aplikovaní na špecifické zariadenie alebo zariadenia.

2.2.2 Politiky služieb a politiky použitia

Politiky služieb [20] popisujú vytváranie služieb na sieti. Organizujú sieťové prostriedky tak, aby poskytli požadované služby, ktoré neskôr môžu pokryť potreby užívateľov. Aplikovaním týchto politík služieb vznikne jeden alebo viacero pomenovaných objektov, ktoré reprezentujú sieťové služby a môžu sa aplikovať na politiky použitia. *Politiky použitia* [20] popisujú, ako alokovať služby vytvorené politikami služieb. Politiky použitia popisujú konkrétny mechanizmus, ktorý sa používa na udržanie súčasného stavu objektu alebo na prechod objektu zo súčasného stavu do nového stavu za účelom využitia služby.

2.3 YANG

Network Configuration Protocol

Network Configuration Protocol (Siet'ový konfiguračný protokol, NETCONF) [7] je mechanizmus, ktorý umožňuje spravovat' siet'ové zariadenia, získavat' konfiguračné dáta a nahrávat', či upravovat' novú konfiguráciu. Tento protokol umožňuje pristupovat' k úplnému rozhraniu pre programovanie aplikácie (API). Aplikácia môže používat' toto API na čiastočný alebo úplný prístup ku konfiguračným dátam. NETCONF používa ako spôsob komunikácie medzi klientom a serverom vzdialené volanie procedúr (RPC). Klient zakóduje RPC do XML[2] a pošle ho serveru. Klientom môže byt' skript alebo aplikácia, server je zvyčajne siet'ové zariadenie. *NETCONF relácia* [7] je logické spojenie medzi siet'ovým administrátorom alebo aplikáciou na správu siete a siet'ovým zariadením.

Oddelenie konfigurácie a stavových dát

Informácie, ktoré sa dajú získat' zo systému v prevádzke, sa dajú rozdelit' do dvoch tried, konfiguračné dáta a stavové dáta. Konfiguračné dáta sú množina zapisovateľných dát, ktorá je potrebná na tranformáciu systému z jeho východiskového stavu do súčasného stavu. Stavové dáta sú doplnkové dáta systému, ktoré nie sú konfiguračné dáta, napr. štatistiky alebo stavové informácie určené len na čítanie. NETCONF rozlišuje medzi týmito typmi dát a poskytuje operácie pre každý z nich: `<get-config>` na získanie len konfiguračných dát a `<get>` na získanie konfiguračných aj stavových dát. [7]

Požiadavky na transportný protokol

NETCONF používa komunikáciu založenú na RPC. Klient pošle postupnost' RPC správ, na ktoré server odpovedá zodpovedajúcou postupnost' RPC správ. NETCONF môže využívat' l'ubovoľný protokol, ktorý poskytuje požadovanú funkcionalitu. Nie je naviazaný na žiadny konkrétny transportný protokol, ale je možné namapovat' ho. Transportný protokol ale musí poskytovat' spôsob, akým indikovat' typ relácie (klient alebo server) vrstve NETCONF protokolu. Okrem orientácie na spojenie, musí transportný protokol poskytovat' autentifikáciu, dátovú integritu, dôveryhodnost' a ochranu proti útoku prehrávaním. [7]

Jazyk YANG

YANG [5] je jazyk na modelovanie dát, ktorý sa používa na modelovanie konfiguračných a stavových dát, ktoré sú spracovávané NETCONF protokolom. YANG sa používa na modelovanie operácií a obsahu NETCONFu.

Funkčný popis

YANG je jazyk na modelovanie dát pre NETCONF protokol. YANG modeluje hierarchickú organizáciu dát ako strom, v ktorom ma každý uzol meno a buď hodnotu alebo množinu potomkov. YANG poskytuje jasný a stručný popis uzlov, rovnako ako aj interakcií medzi nimi.[5]

YANG štrukturuje dátové modely do modulov a podmodulov. Modul obsahuje tri typy príkazov: hlavička modulu, revízia a definícia. Príkaz *hlavička modulu* popisuje modul a

[2]Extensible Markup Language [`http://www.w3.org/TR/2000/REC-xml-20001006`]

poskytuje informácie o module, príkaz *revízia* poskytuje informácie o histórii modulu a príkazy *definícia* sú telom modulu, definujú dátový model. NETCONF server môže mať implementovaných niekoľko modulov, ktoré umožňujú rôzne pohľady na rovnaké dáta alebo pohľady na disjunktné podsekcie dát zariadenia. No takisto môže server implementovať len jeden modul, ktorý definuje všetky dostupné dáta. Samotný modul môže byť rozdelený do podmodulov, no navonok sa bude javiť ako celok. Ďalej môže importovať dáta z externých modulov a vkladať dáta z podmodulov.[5]

YANG je jazyk na rozhraní medzi vysokoúrovňovým a nízkoúrovňovým popisom. Pri čítaní modulu napísaného v jazyku YANG je vidieť, ako sú dáta kódované na NETCONF operácie.[5]

Ide o rozšíriteľný jazyk, rozšírenia môžu byť definované výrobcami ako aj jednotlivcami. Syntax príkazov jazyku YANG umožňuje koexistenciu rozšírení so štandardnými príkazmi, no zároveň je možné postrehnúť rozšírenia v module.[5]

YANG obmedzuje rozsah problémov, ktoré rieši, na dátové modely protokolu NETCONF, nie na ľubovoľné XML dokumenty. Aby rozšíril už existujúce riešenia, udržiava kompatibilitu s protokolom SMNP[3], SMIv2[4].[5]

Základné modelovanie dát

V jazyku YANG sú definované štyri typy uzlov na modelovanie dát. [5]
Listové uzly [5], ktoré obsahujú jednoduché dáta, ako celočíselné hodnoty alebo reťazce. Tento typ uzlu má práve jednu hodnotu a žiadnych potomkov. Príkladom je uzol definujúci meno hostiteľa [5]:

```
leaf host-name {
    type string;
    description "Hostname of this device";
}
```

Zoznamy listov [5] sú sekvencie listových uzlov práve jedného konkrétneho typu, napríklad [5]

```
leaf-list domainsearch
    type string;
    description "List of domain names";
```

Kontajnery [5] sa používajú na zoskupenie súvisiacich uzlov do podstromu. Kontajner má len potomkov a nemá žiadnu hodnotu. Môžu obsahovať potomkov akéhokoľvek typu. Napríklad [5]

```
container system {
    containter login {
        leaf message {
            type string;
     description "Message on the startup";
     }
     }
```

[3]Simple Network Management Protocol – `https://www.ietf.org/rfc/rfc1157.txt`
[4]Structure od Management Information version 2 – `https://tools.ietf.org/html/rfc2578`

```
    }
}
```

Zoznamy [5] sú uzly, ktoré definujú postupnosti zoznamov. Každá položka je ako štruktúra a je unikátne identifikovaná hodnotami listových uzlov. Zoznam môže definovať viaceré kľúčové listy a môže obsahovať ľubovoľný počet potomkov akéhokoľvek typu. Príkladom je [5]

```
list user {
    key "name";
    leaf name {
        type string;
    }
    leaf full-name {
        type string;
    }
    leaf class {
        type string;
    }
}
```

Stavové dáta

YANG [5] dokáže modelovať tiež stavové dáta. Keď je uzol označený ako `config false`, jeho podhierarchia je označená ako stavové dáta.

Vstavané typy

Jazyk YANG obsahuje vstavané typy [5], podobné k tým z programovacích jazykov, ale s určitými rozdielmi vzhľadom na potreby manažovania konfigurácie. Tabuľka 2.1 ich sumarizuje.

Odvodené typy

YANG umožňuje definovať odvodené typy [5] zo základných typov použitím príkazu `typedef`. Základný typ môže byť buď vstavaný typ, alebo odvodený typ, podľa hierarchie odvodených typov. Napríklad [5]:

```
typedef percent {
    type uint8 {
       range "0 .. 100";
    }
    description "Percentage";
}
leaf completed {
type percent;
}
```

Jazyk YANG okrem týchto základných funkcií umožňuje vytvárať znovupoužiteľné skupiny uzlov (napríklad zoskupenie cieľovej IP adresy a cieľového portu), vylúčenie nekompatibilných uzlov na základe prípadov uvedených pre rôzne možnosti, rozšírenie dátového modelu, definovanie nových vzdialene volaných procedúr a notifikácií.[5]

Tabulka 2.1: Vstavané typy v jazyku YANG

Meno	**Popis**
binary	akékoľvek binárne dáta
bits	množina bitov alebo príznakov
boolean	„true“ alebo „false“
decimal64	64-bitové znamienkové dekadické číslo
empty	list bez hodnoty
enumeration	vymenované ret'azce
identityref	referencia na abstraktnú identitu
instance-identifier	referencie na dátový uzol stromu
int8	8-bitové znamienkové celé číslo
int16	16-bitové znamienkové celé číslo
int32	32-bitové znamienkové celé číslo
int64	64-bitové znamienkové celé číslo
leafref	referencia na inštanciu listu
string	ľud'mi čitateľný ret'azec
uint8	8-bitové bezznamienkové celé číslo
uint16	16-bitové bezznamienkové celé číslo
uint32	32-bitové bezznamienkové celé číslo
uint64	64-bitové bezznamienkové celé číslo
union	výber z príslušných typov

Kapitola 3

Jazyk na definíciu sieťových politík

Väčšina konfigurácií sieťových zariadení je určitým spôsobom hierarchicky štrukturovaná. Zariadenia firmy Cisco [26] či Juniper [27] používajú stromovú štruktúru. Podobný prístup používa aj modelovací jazyk YANG, ktorý popisuje dátový model využívaný protokolom NETCONF. Preto bola zvolená schéma jazyka reflektujúca stromovú štruktúru.

Každá politika obsahuje informácie o tom, ktoré vlastnosti sa budú konfigurovať, na ktorých zariadeniach sa majú konfigurovať a aké podmienky musia byť dodržané, aby bola konfigurácia validná. Jazyk, ktorý to popisuje, pozostáva z dvoch častí. Dáta vo formáte JSON[1] obsahujú informácie potrebné pre nakonfigurovanie požadovaných vlastností a zoznam zariadení, kde sa tieto nastavenia majú aplikovať. Či je to možné, vyhodnotia podmienky v zdrojovom kóde programu.

3.1 Časť vo formáte JSON

Politiku popísanú vo formáte JSON je možné vo všeobecnosti popísať nasledovne:

```
{
    <meno>: <názov politiky>,
    <zoznam zariadení>: {
        <zariadenie 1>: {
            <parameter 1>: <hodnota 1>,
            <parameter 2>: <hodnota 2>
            ...
            },
        <zariadenie 2>: {
            <parameter 1>: <hodnota 3>,
            <parameter 2>: <hodnota 4>
            ...
        }
    },
    <parameter 3>: <hodnota 5>
    ...
}
```

[1] http://www.json.org/

Položka `názov politiky` je meno politiky, ktoré určuje základné správanie programu na konfiguráciu. Ďalšou položkou sú názvy zariadení, teda `zariadenie 1` , `zariadenie 1`, atď. Na základe nich sa vyberú zariadenia, na ktorých sa táto politika aplikuje. Kombinácia `parameter x` a `hodnota y` určujú, aké hodnoty budú na ovplyvnených zariadeniach nastavené. V závislosti od typu politiky môžu byť nastavované pre každé zariadenie zvlášť (`parameter 1` a `parameter 2`) alebo pre všetky zariadenia vymenované v politike (`parameter 3`).

3.2 Podmienky politiky popísané v kóde

Táto časť popisuje obmedzenia politiky, teda podmienky, ktoré musia byť splnené, aby bolo možné aplikovať túto politiku. Ide o overenie, či sú pre parametre nastavené validné hodnoty, alebo či sa jednotlivé parametre, resp. ich hodnoty vzájomne nevylučujú. Týmto spôsobom sa overí, či pri nastavovaní IP adries na dvoch rozhraniach sú IP adresy pre obe rozhrania z rovnakej siete, alebo či sa pri smerovacom protokole RIP [26] nenastavuje autentifikácia pre verziu 1.

Ide o obmedzenia politík popísané ako podmienky v jazyku Python. Nakoľko každá politika vyžaduje iné kritéria, existujú rôzne množiny obmedzení, teda rôzne podmienky v kóde. Napr.:

```
ipsOK = False
netOK = False
maskOK = False
ipNotSame = False
unique = False

setInterfaces(ipadd1, int1, hostname1, ipadd2, int2, hostname2)
ipNotSame = checkSameIP(str(ipint1), str(ipint2))
ipsOK = checkIPs()
maskOK = checkMask()
unique = checkIfUnique()
netOK = checkNetwork()

if (ipsOK and netOK and maskOK and ipNotSame and unique):
    print "EVERYTHING OK"
else:
    print "There was an error in configuration."
    print "Please correct it and try again."
    return False
```

Vyššie je zobrazená časť kódu overujúca niekoľko podmienok, ktoré musia byť splnené, aby sa mohlo vytvoriť spojenie medzi zariadeniami. V tomto prípade sú pre jednotlivé podmienky volané separátne funkcie, v ktorých sa jednotlivé podmienky vyhodnotia. V inom prípade, ako je napr. konfigurácia smerovacieho protokolu RIP, sa tok programu riadi práve podľa odmedzení. Nasledujúci kód popisuje závislosť možnosti nastaviť autentifikáciu od použitej verzie protokolu:

```
 authentication = False
    if "key" in configuration:
```

```
if version == "1":
    print "RIPv1 does not support authentication."
    print "Please correct your config and try again."
    return False
else:
    authentication = True
```

Kapitola 4

Nástroje na syntézu a sieť ové modely na reprezentáciu sieť ovej konfigurácie

4.1 Nástroje na syntézu konfigurácie

4.1.1 ConfigAssure

ConfigAssure [15] je nástroj, ktorý umožňuje preklenúť medzru medzi požiadavkami infraštruktúry a detailnou konfiguráciou, ktorá implementuje tieto požiadavky. Základom nástroja ConfigAssure je *Requirement Solver*. Vstupmi preň sú požiadavky a konfiguračná databáza. Množiny v konfiguračnej databáze môžu obsahovať konfiguračné premenné. Solver sa pokúša vypočítať hodnoty pre tieto premenné, také, aby po nahradení v databáze spĺňala databáza požiadavky. Ak neuspeje, dokazuje nesplniteľnosť požiadaviek. Requirement Solver rieši nasledujúce fundamentálne problémy [15]:

- Špecifikácia – všetky požiadavky sú vyjadrené ako obmedzenia konfigurácie. Obmedzenie je zjednocujúci koncept, kde unifikárotom je dané obmedzenie, ktoré predstavuje požiadavky na konfiguráciu. Požiadavky na bezpečnosť, funkcionalitu, výkon a spoľahlivosť môžu byť považované za obmedzenia. Konfiguračné informácie sú zvyčajne reprezentované databázou. Význam požiadaviek je definovaný čiastkovým vyhodnotením, ktoré spočíta ekvivalent ako formulu bez kvantifikátorov.

- Syntéza konfigurácie – deklaratívna povaha nástroja Requirement Solver poskytuje výhodu kompozičnosti. Ak existujú požiadavky A a B, Requirement Solver spočíta $\mathtt{A} \wedge \mathtt{B}$, teda riešenie vyhovuje podmienkam A aj B.

- Diagnostika konfiguračnej chyby – pokiaľ je požiadavka nesplniteľná, Solver vypočíta dôkaz nesplniteľnosti ako množinu jednoduchých obmedzení, ktoré sú takisto neriešiteľné. Ak táto množina obsahuje obmedzenie vo forme $\mathtt{x} = \mathtt{c}$, kde $\mathtt{x}$ je konfiguračný parameter a $\mathtt{c}$ je konštanta, potom $\mathtt{x} = \mathtt{c}$ je platná hlavná príčina. Určuje, ktorý konfiguračný parameter prispieva k neriešiteľnosti.

- Opravenie konfiguračnej chyby – Ak sa obmedzenie $\mathtt{x} = \mathtt{c}$ objaví v dôkaze neriešiteľnosti a takisto aj v pôvodných požiadavkách, odstránenie tohoto obmedzenia je dobrá heuristika pre obnovenie riešiteľnosti. Oprava problému však môže narušiť ostatné

požiadavky, preto je nutné, aby zmena vyhovovala všetkým požiadavkam, nielen tej problematickej.

Requirement Solver je implemenovaný ako Kodkod[1], nástroj na hľadanie modelov. Tento nástroj umožňuje špecifikovať obmedzenia logiky prvého rádu v konečnej množine, transformovať ich na boolean formuly, tieto riešiť pomocou nástrojov na riešenie SAT[2] a následne reflektovať výsledky naspäť do modelu logiky prvého rádu. Pokiaľ to nie je možné, dokáže neriešiteľnosť. Kodkod hrá kľúčovú úlohu ako efektívny a škáľovateľný nástroj na riešenie SAT problému. Akceptuje QFF[3] priamo na vstupe, poskytuje riešenia pre dôkaz neriešiteľnosti a vďaka optimalizáciam redukuje množstvo generovaných boolean obmedzení. Napriek sile nástroja Kodkod, veľa častí požiadaviek môže byť vyriešených efektívnejšie špecializovanými nástrojmi na riešenie obmedzení, databázovými jadrami (engine) alebo algoritmami, ktoré využívajú čiastkové informácie dostupné v konfiguračnej databáze. Vyriešenie týchto častí môže zredukovať počet obmedzení, ktoré na vyriešenie potrebujú možnosti nástroja na hľadanie modelu. [15]
ConfigAssure [15] plní tento plán implemenovaním *formuly bez kvantifikátorov*, QFF, ktorá pozostáva z booleanovských kombinácií jednoduchých aritmetických obmedzení nad celými číslami. Požiadavka je špecifikovaná definovaním čiastkového vyhodnotenia, ktoré ju transformuje na ekvivalentnú QFF. Dôležité je vynechať z QFF obmedzenia, ktoré môžu byť vyriešené inak ako hľadaním modelu. Potom sa QFF pošle do Kodkodu na vyriešenie. Nástroj na čiastkové vyhodnocovanie (partial evaluator) pokrýva všetky ostatné oblasti záujmu v danej doméne. QFF majú viaceré výhody [15]:

- ich vysokoúrovňová povaha zjednodušuje dizajn nástroja na čiastkové vyhodnocovanie,
- môžu byť efektívne riešené Kodkodom,
- dôkazy ich neriešiteľnosti zjednodušujú algoritmy na diagnostiku a opravenie chýb.

Requirement Solver – nástroj na riešenie požiadaviek

Primitíva

Predpokladáme, že existuje nespočetná, nekonečná množina konfiguračných premenných, funkčných symbolov, predikátových symbolov a symbolov konfiguračnej databázy so žiadnym alebo viacerými parametrami. Funkcie bez argumentov sa nazývajú skaláry. Množina skalárov zahŕňa aj celé čísla. Requirement Solver pracuje pre konkrétnu špecifikáciu s konečným počtom premenných a symbolov z vyššie uvedenej množiny. Príkladom je zoznam hostiteľských mien. [15]

Termy

Každá konfiguračná premenná a každý skalár je term. Ak $\mathtt{x}_1, ..., \mathtt{x}_k$ je term a $\mathtt{F}$ je funkčný symbol s $\mathtt{k}$ parametrami, potom $\mathtt{F}(\mathtt{x}_1, ..., \mathtt{x}_k)$ je term. [15]

[1] `http://alloy.mit.edu/kodkod/`
[2] satisfiability – splniteľnosť
[3] Quantifier-free form – formula bez kvantifikátorov

Konfiguračná databáza

Ak P je databázový symbol s k parametrami a $x_1, ..., x_k$ je buď skalár aleob konfiguračná premenná, potom $P(x_1, ..., x_k)$ je databázová n-tica. [15]

Požiadavky

Ak $x_1, ..., x_k$ sú termy a R je predikátový symbol s k parametrami, potom $R(x_1, ..., x_k)$ je požiadavka. Ak R_1 a R_2 sú požiadavky, potom aj $not(R_1)$, $and(R_1, R_2)$, $or(R_1, R_2)$ a $implies(R_1, R_2)$ sú požiadavky. Požiadavky nemajú expolicitný kvantifikátor, ale môžu nadobudnúť význam zhodný s formulou predikátovej logiky prvého rádu. Požiadavka bez konfiguračnej premennej je buď platná alebo neplatná nad databázou bez konfiguračných premenných. [15]

Formula bez kvantifikátorov – QFF

QFF je obmedzenie vytvorené z konfiguračných premenných , celých čísel, funkčných symbolov pre ščítanie, odčítanie a bitové operácie a z predikátových symbolov =, <, >, >=, <=. [15]

Partial Evaluator – nástroj na čiastkové vyhodnocovanie

Tento nástroj definuje význam požiadaviek v kontexte databázy. Transformuje požiadavky na ekvivalentné QFF. Nech σ je priradenie premenných k celým číslam $\{< x_1 = v_1 >, ..., < x_k = v_k >\}$, každé x_i je konfiguračná premenná a každé v_i je celé číslo. Nech Req je požiadavka, DB databáza a C QFF. Nech $Req\sigma$ je výsledok nahradenia každej premennej v Req jej hodnotou v σ. Podobne pre $DB\sigma$ a $C\sigma$. Teraz $eval(Req, DB, C)$ znamená, že pre každé priradenie σ, $Req\sigma$ je pravdivé al $DB\sigma$ je ekvivalentné s $C\sigma$. Partial Evaluator pre konfiguračnú aplikáciu je definícia eval predikátu pre všetky požiadavky a databázy, ktoré sú podstatné pre danú aplikáciu. Podstatným princípom v implementácii $eval(Req, DB, C)$ je, že C nemá obsahovať obmedzenie, ktoré je možné vyhodnotiť inak ako hľadaním modelu. [15]

Rozhranie Kodkodu

Toto rozhranie je definované predikátom Prologu[4] $solve(Q, Result)$, kde Q je QFF. Ak je Q riešiteľná, Result je term $solvable : \sigma$, kde σ je priradenie konfiguračných premenných k celým číslam tak, aby splnili Q. Ak je Q nesplniteľné, Result je term $unsolvable : P$, kde P je dôkaz neriešiteľnosti. Tento dôkaz je zoznam formúl bez kvantifikátorov, ktorých konjunkcia je neriešiteľná. [15]

Riešenie základných konfiguračných problémov

Požiadavku je možné špecifikovať definovaním jej čiastkového vyhodnotenie (eval). Nie je nutné definovať eval pre každú požiadavku, je možné použiť šablóny. Pre určitú aplikačnú doménu existuje Requirement Library (knižnica), ktorá obsahuje procedúry na efektívne riešenie základných typov požiadaviek v danej doméne. Ich kombinácia umožňuje definovať komplexné požiadavky.
Pri syntéze konfigurácie pre danú požiadavku R a konfiguračnú databázu DB, aby sa našla

[4] Programming in Logic – napr. http://www.swi-prolog.org/

premenná σ taká, že $\texttt{R}\sigma$ platí nad $\texttt{DB}\sigma$, sa použije nasledovná otázka v Prologu: $\texttt{eval}(\texttt{R}, \texttt{DB}, \texttt{Q})$, $\texttt{solve}(\texttt{Q}, \texttt{solvable} : \texttt{X})$. Ak otázka uspeje, bude premenná $\texttt{X}$ zviazaná s σ.[15]

Pre diagnostiku konfiguračných chýb, nech $\texttt{DB}$ je konfiguračná databáza obsahujúca premenné $\texttt{x}_1, ..., \texttt{x}_k$. Nech $\texttt{Relaxable}$ je konjunkcia primitívnych obmedzení $\texttt{x}_1 = \texttt{v}_1, ..., \texttt{x}_k = \texttt{v}_k$, kde každé $\texttt{x}_i$ je konfiguračná premenná a každé $\texttt{v}_i$ je celočíselná hodnota. $\texttt{Relaxable}$ sa okrem špecifikovania počiatočných hodnôt množiny premenných používa aj na indikovanie, že tieto hodnoty môžu byt' v prípade potreby uvoľnené. Predpokladajme, že pre nejaké $\texttt{Req}$, $\texttt{DB}$ a $\texttt{Q}$, $\texttt{eval}(\texttt{Req}, \texttt{DB}, \texttt{Q})$ je $\texttt{and}(\texttt{Q}, \texttt{Relaxable})$ neriešiteľné. Potom otázka na Prolog $\texttt{eval}(\texttt{Req}, \texttt{DB}, \texttt{Q}), \texttt{solve}(\texttt{and}(\texttt{Q}, \texttt{Relaxable}), \texttt{unsolvable} : \texttt{Proof})$ uspeje, zviaže premennú Prologu $\texttt{Proof}$ so zoznamom QFF, ktorých konjunkcia je takisto neriešiteľná. Pokiaľ obmedzenie $\texttt{x}_i = \texttt{v}_i$ je súčast'ou zoznamu $\texttt{Proof}$, stane sa dôvodom neriešiteľnosti danej požiadavky. Ak takéto obmedzenie neexistuje, algoritmus zastaví.[15]

Cieľom pri opravovaní konfiguračnej chyby je nájst' alternatívne $\texttt{x}_i$, odstránit' $\texttt{x}_i = \texttt{v}_i$ z $\texttt{Relaxable}$ a vytvorit' tak $\texttt{Relaxable}'$ a pokúsit' sa vyriešit' požiadavku pre novú hodnotu: $\texttt{and}(\texttt{Q}, \texttt{Relaxable}')$. Ak $\texttt{solve}$ uspeje, nájde novú hodnotu $\texttt{x}_i$ a upraví hodnotu $\texttt{v}_i$. Pokiaľ neuspeje, znovu vypočíta dôkaz neriešiteľnosti a postup opakuje. Ak ale takéto obmedzenie neexistuje, algoritmus zastaví.[15]

4.2 Siet'ové modely

Siet'ová konfigurácia je konečná množina príkazov. Môže byt' už existujúca, teda tá, ktorá je v danom momente aktívna na siet'ovom zariadení, alebo nová, ktorá vytvára alebo modifikuje existujúcu. Ak nemá byt' vytváranie siet'ovej konfigurácie zariadenia alebo jej zmena manuálnou prácou siet'ového administrátora, ale má byt' výstupom nástroja na správu siete, je potrebné previest' konfiguráciu siet'ového zariadenia do abstraktnejšej formy, ktorá vyhovuje správcovskému nástroju. Teda vytvorit' jej model.

4.2.1 Existujúce modely

Vzhľadom na povahu nástroja na správu či generovanie sieťovej konfigurácie, existujú viaceré modely. Môže ísť o abstraktné dátové štruktúry, alebo viac formálne riešenia ako napr. predikátová logika či graf. Jazyk na popis siet'ovej konfigurácie YANG [5] používa ako model *strom*. Jeho hierarchia rovnako ako aj základné konštrukcie a operácie sú súčast'ou popisu jazyka v 2.3. Na druhú stranu, *predikátová logika* sa používa v nástroji ConfigAssure [15], ktorý je schopný syntetizovat' siet'ovú konfiguráciu. Transformuje konfiguračné príkazy na formuly logiky prvého rádu, čo mu umožňuje detegovat' a príp. opravovat' konfiguračné konflikty. Tento model je popísaný v 4.1.1. Rovnaký prístup, avšak s použitím iného jazyku, bol použitý aj v [14]. Ďalšie použité modely sú *graf* a *stavový automat*.

Graf

V [17], kde autori popisujú nástroj na automatickú syntézu konfigurácie siet'ovej bezpečnosti, je ako model siete použitý graf. Tento model je definovaný ako $\langle \mathbb{N}, \mathbb{L} \rangle$, kde $\mathbb{N}$ definuje konečnú množinu siet'ových uzlov, vrátane smerovačov a hostiteľov. Hostiteľ môže poskytovat' jednu alebo viacero služieb, ku ktorým pristupujú ostatní hostitelia. Služba je označná ako $g \in \mathbb{G}$, kde $\mathbb{G}$ je množina všetkých služieb. Označenie $g(i, j)$ definuje dátový tok medzi dvojicou hostiteľov $\{i, j\}$, kde i je zdrojom a j cieľom v rámci služby g. $\mathbb{L} \subseteq \mathbb{N} \times \mathbb{N}$ je konečná množina liniek, ktoré definujú prepojenia medzi siet'ovými hostiteľmi.

Stavový automat

Autori [1] popisujú model siete prostredníctvom obrovského konečného stavového automatu. Stav siete určujú pakety. Informácie relevantné pre tento stavový automat sa nachádzajú v hlavičke paketu. Ďalšou podstatnou informáciou je lokalita, kde sa paket nachádza. V zjednodušenom modeli sa neberie do úvahy obsah dátovej časti paketu.

Jediné informácie, ktoré sú potrebné v zjednodušenom modeli sú zdrojová a cieľová adresa, zdrojový a cieľový port a aktuálna lokalita, kde sa paket nachádza. Stav siete je možné zakódovať nasledujúcou charakteristickou funkciou:

$$\sigma : \mathtt{IP_s} \times \mathtt{port_s} \times \mathtt{IP_d} \times \mathtt{port_d} \times \mathtt{loc} \rightarrow \{\mathtt{true}, \mathtt{false}\}$$

$\mathbf{IP}_s$	32-bitová zdrojová IP adresa
$\mathbf{port}_s$	16-bitový zdrojový port
$\mathbf{IP}_d$	32-bitová cieľová IP adresa
$\mathbf{port}_d$	16-bitový cieľový port
loc	32-bitová IP adresa zariadenia, ktoré spracováva paket v danom momente

Funkcia σ sa vyhodnotí ako pravdivá, pokiaľ vstupné parametre zodpovedajú paketu, ktorý je na sieti, v opačnom prípade sa vyhodnotí ako nepravdivá. Každé zariadenie môže byť modelované na základe toho, ako pristupuje k paketu. Napríklad firewall môže prepustiť paket, no takisto ho môže zo siete odstrániť, smerovač môže zmeniť lokalitu paketu a zariadenie na prekladanie sieťových adries môže zmeniť tieto informácie v hlavičke. Správanie týchto zariadení je popísané zoznamom pravidiel, pričom každé pravidlo pozostáva z podmienky a akcie. Podmienky môžu byť popísané booleanskými formulami nad stavovými bitmi. Ak paket v zariadení splní podmienku pravidla, vykoná sa zodpovedajúca akcia.[1]

V rozšírenom modeli sa berie do úvahy takisto obsah dátovej časti paketu, teda je potrebné rozšíriť stavový priestor. Je potrebné vyhodnocovať aj informácie z hlavičky IP, ktoré sa môžu nachádzať v tele paketu. V tom prípade sa pridajú ďalšie kópie týchto polí a tiež kópia pre každú úroveň zapúzdrenia.[1]

Kapitola 5

Reprezentácia topológie

5.1 Graf

Model siete je abstrakciou nad existujúcimi spojeniami na vrstve L3[1] (alebo aj L2 v prípade siet'ových prepínačov) vo fyzickej siet'ovej topológií. Základné informácie, ktoré musí obsahovat', sú informácie o zariadeniach a o spojeniach medzi týmito zariadeniami. Jednou z vhodných štruktúr na reprezentáciu požadovaných informácií je graf.

Z matematického hľadiska existujú *neorientované* a *orientované* grafy. Neorientovaný graf $\mathtt{G}$ sa skladá z množiny vrcholov $\mathtt{V}$ a množiny hrán $\mathtt{H}$ tak, že každá hrana je priradená neusporiadanej dvojici vrcholov $\mathtt{u}, \mathtt{v} \in \mathtt{V}$. Ak existuje jediná hrana $\mathtt{h} \in \mathtt{H}$ priradená dvojici vrcholov $\mathtt{u}, \mathtt{v} \in \mathtt{V}$, píšeme $\mathtt{h} \equiv \{\mathtt{u}, \mathtt{v}\}$. Vo všeobecnosti môže byt' jednej dvojici vrcholov priradných viacero hrán, tieto hrany sa nazývajú násobné. Podobne orientovaný graf $\mathtt{G}$ sa skladá z množiny vrcholov $\mathtt{V}$ a množiny hrán $\mathtt{H}$ tak, že každej hrane $\mathtt{h} \in \mathtt{H}$ je priradená usporiadaná dvojica $\mathtt{u}, \mathtt{v} \in \mathtt{V} \times \mathtt{V}$ vrcholov $\mathtt{u}, \mathtt{v} \in \mathtt{V}$. Ak existuje jediná hrana $\mathtt{h} \in \mathtt{H}$ priradená dvojici vrcholov $\mathtt{u}, \mathtt{v} \in \mathtt{V}$, píšeme $\mathtt{h} \equiv \{\mathtt{u}, \mathtt{v}\}$.

V prípade siet'ovej topológie zariadenia tvoria množinu vrcholov $\mathtt{V}$ a spojenia medzi nimi množinu hrán $\mathtt{H}$. Keďže ide o spojenia na vrstve L3, používa sa v tejto implementácii neorientovaný graf.

5.2 Implementácia

V jazyku Python existuje dátová štruktúra slovník (dictionary), nazývaná aj asociatívne pole, ktorá je indexovaná pomocou kľúčov [28]. Vďaka tomu je veľmi vhodná na reprezentáciu grafu, teda siet'ovej topológie. Príklad takejto reprezentácie je zobrazený nižšie.

```
{
  'A': ['B', 'C'],
  'B': ['A', 'C'],
  'C': ['A', 'B']
  }
```

Graf je reprezentovaný ako slovník, kde kľúčom je názov zariadenia a hodnotou je pole obsahujúce názvy zariadení, ktoré sú spojené zo zariadením, ktoré bolo kľúčom. Aby bolo možné dohľadat' použité rozhrania na týchto spojeniach, existuje pole polí dvojíc, kde

[1] http://www.itu.int/rec/T-REC-X.200-199407-I

dvojicu tvorí názov zariadenia a rozhranie na tomto zariadení a spojenie medzi dvomi zariadeniami reprezentuje práve pole:

```
[
  [
    ['A', 'GigabitEthernet0/3'],
    ['B', 'GigabitEthernet0/2']
  ],
  [
    ['A', 'GigabitEthernet0/2'],
    ['C', 'GigabitEthernet0/3']
  ],
  [
    ['B', 'GigabitEthernet0/3'],
    ['C', 'GigabitEthernet0/2']
   ]
]
```

Pre potreby programu na vygenerovanie a aplikovanie sieťovej konfigurácie je potrebné uchovávať informácie o názve zariadanie a jeho IP adrese. Pre realizáciu tejto požiadavky bol použitý taktiež slovník, pričom kľúčom sú názvy zariadení a hodnotou IP adresa.

```
{
  'A': '172.16.0.88',
  'B': '172.16.0.89',
  'C': '172.16.0.90',
}
```

5.3 Inicializácia

Aby bolo možné využívať dátové štruktúry popísané v 5.2, je nutné ich najprv inicializovať. Údaje popisujúce zariadenia a prepojenia medzi nimi sa pri štarte programu načítajú do pamäte. Sú uložené v súbore vo formáte JSON.

Kapitola 6

Architektúra a použité systémy

Na komunikáciu so sieťovými zariadeniami bol vybraný protokol NETCONF, ktorého stručná charakteristika je uvedená v kapitole 2.3. Podľa [8] je možné vytvoriť SSH[1] spojenie medzi počítačom a zariadením a týmto spojením zasielať a prijímať správy protokolu NETCONF. Túto komunikáciu zabezpečuje utilita *ncclient* [3] a program, ktorý je predmetom tejto práce.

Keďže program na konfiguráciu zariadení využíva pripojenie cez SSH (prostredníctvom programu *ncclient*), potrebuje poznať prístupové údaje k zariadeniam. Tieto informácie berie zo súboru `access.json`, kde môžu byť nastavené rovnaké prístupové údaje (meno a heslo) pre všetky zariadenia alebo pre každé zariadenie zvlášť.

Program po spustení načíta konfiguráciu topológie zo vstupného súboru, ktorý sa mu zadá ako argument. Tento súbor obsahuje informácie o sieťových zariadeniach v použitej topológii a o existujúcich spojeniach medzi týmito zariadeniami. Na základe týchto informácií sa vygeneruje graf reprezentujúci sieťovú topológiu, tak ako je popísané v kapitole 5

Tento program ďalej načíta zo súboru vstupnú konfiguráciu, ktorá obsahuje informácie o sieťových zariadeniach v aktuálnej topológii. Následne sa pomocou utility *ncclient* stiahnu aktuálne konfigurácie zo zariadení. Ďalej sú spracované a uložené v programe. Konfigurácie zo zariadení sú ukladané v priečinku `config` a v prípade, že užívateľ chce vrátiť vykonané zmeny, môže túto konfiguráciu nahrať manuálne.

Ďalší beh programu závisí od politiky, ktorá sa aplikuje. Každý z nich zahŕňa získanie vstupných informácií, volanie určitej postupnosti príkazov utility *ncclient*, volanie metód z tried alebo funkcií z modulov, ktoré vytvárajú príkazy pre *ncclient* alebo spracovávajú správy protokolu NETCONF získané touto utilitou, a ktoré obsahujú určité mechanizmy na overenie správnosti konfigurácie.

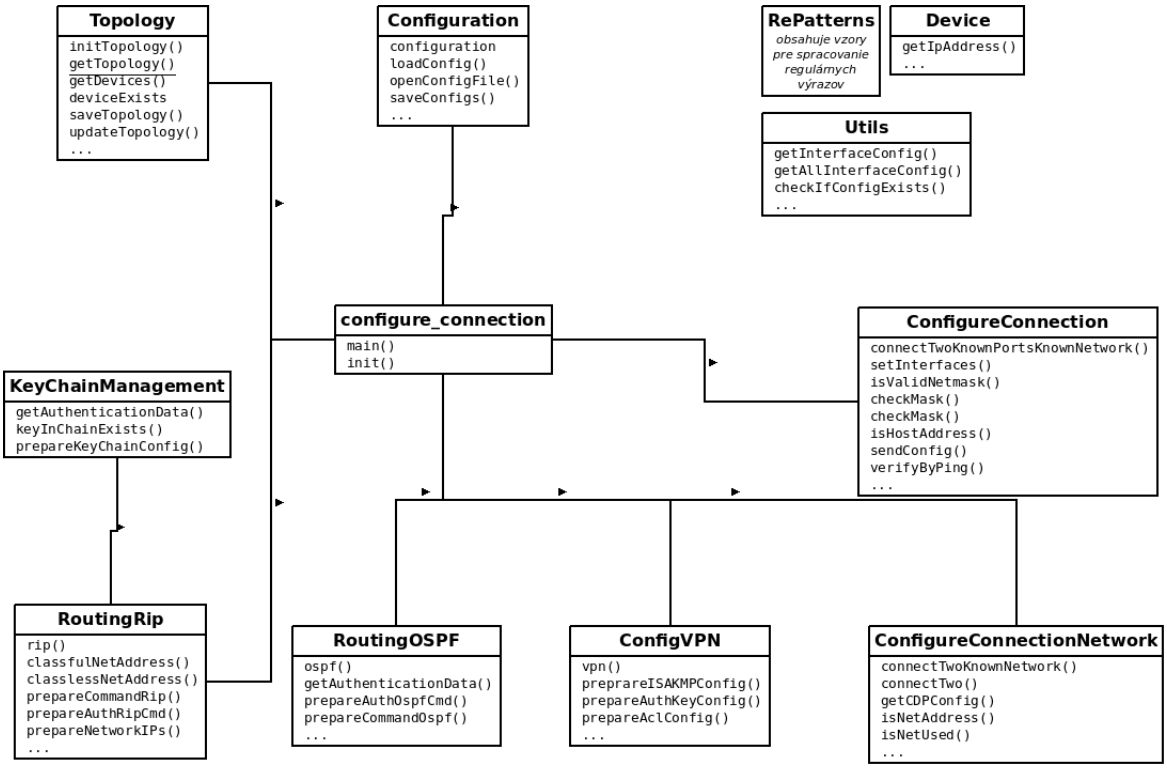
Po úspešnom skončení, program uloží aktuálny stav topológie, teda zoznam zariadení a spojenia medzi nimi do výstupného súboru `topology_out.json`.

6.1 Architektúra

Program spracovávajúci a generujúci sieťovú konfiguráciu je napísaný v programovacom jazyku Python 2.7. Nosnou konštrukciou sú moduly *Topology*, *Device*, konkrétne trieda *Device* a *Connection*, ktoré popisujú a udržujú informácie o sieťovej topológii, zariadeniach,

[1] Secure Shell

resp. spojeniami medzi nimi. Tie sú doplnené pomocnými triedami *IpAddress* a *Interface*, ktoré obsahujú metódy na spracovanie informácií o IP adresách a portoch na zariadeniach.



Obrázek 6.1: Diagram tried a modulov

Na obr. 6.1 sú zobrazené triedy a moduly konfiguračného programu a základné vzťahy medzi nimi. Mmoduly ako *RoutingRip*, *RoutingOSPF*, *ConfigureConnection*, *ConfigureConnectionNetwork* a *ConfigVPN* používajú niektoré funkcie z triedy *Topology* a z modulu *Configuration*. Takisto používajú funkcie z pomocných modulov *Utils*, *RePatterns* a *Device*.

6.2 VIRL

VIRL [29] (Virtual Internet Routing Lab) je virtualizačné prostredie vyvinuté spoločnosťou Cisco. Umožňuje simulovať sieťové topológie zložené zo smerovačov a prepínačov, ktoré používajú IOS, rovnaký operačný systém firmy Cisco, aký sa používa na reálnych sieťových zariadeniach. V tomto prostredí som vytvorila topológie pre simulovanie správania zariadení v jednotlivých scenároch.

Prostredie VIRL beží na virtuálnom počítači. Využíva grafické rozhranie VM Maestro, v ktorom sa navrhne sieťová topológia a spustí sa simulácia. VM Maestro využíva nástroj AutoNetkit na vygenerovanie počiatočnej konfigurácie zariadení. VIRL umožňuje prepojenie simulovanej topológie s existujúcou topológiou pomocou rozhraní *flat*, *flat1* a *SNAT*. Rozhrania *flat* a *flat1* vytvárajú spojenie na vrstve L2, zatiaľ čo *SNAT* sa pripája na vrstve L3.V prostredí VIRL existujú tri módy, ktoré rozlišujú spôsob, akým sú simulované zariadenia pripojené k externému prostrediu:

- privátna sieť pre projekt,
- privátna sieť pre simuláciu,
- zdieľaná *flat* sieť.

Pre privátnu sieť pre projekt alebo pre simuláciu sa vytvorí LXC (Linux container), ktorý „premostuje“ *flat* sieť a sieť určenú na riadenie zariadení (management network). Rozdiel medzi *privátnou sieťou pre projekt* a *privátnou sieťou pre simuláciou* je v tom, že pre prvú sa vytvorí LXC, ktoré má prístup ku všetkým zariadeniam v projekte, a v druhom prípade sa vytvorí separátne LXC pre každú simuláciu, teda má prístup len k zariadeniam v danej simulácii. A v prípade *zdieľanej flat siete* sa nevytvorí žiadny LXC a rozhrania na ovládanie zariadení (management porty) sú umiestnené priamo vo *flat* sieti, teda je možné pristupovať k nim cez toto rozhranie aj mimo prostredia VM Maestro [29]. Tento prístup bol použitý pre prístup konfiguračného programu k sieťovým zariadeniam v tejto práci. V *privátnej sieti pre projekt* sa vytvorí LXC (Linux)

6.3 ncclient

ncclient [3] je knižnica pre Python, určená pre klientskú stranu protokolu NETCONF. Vo verzii 0.5.0 obsahuje už aj podporu pre Python 3, no v tejto práci bola použitá stabilná verzia 0.4.7 pre Python 2.7. *ncclient* poskytuje API, ktoré mapuje XML zápis na prostriedky jazyku Python. Zároveň umožňuje relatívne jednoduchú komunikáciu so zariadeniami práve cez protokol NETCONF. Podľa [4] formát správy na získanie konfigurácie zo zariadenia je nasledujúci:

```
def demo(host, user, expr):
    with manager.connect(host=host, port=22, username=user) as m:
        assert(":xpath" in m.server_capabilities)
        c = m.get_config(source='running', filter=('xpath', expr)).data_xml
        with open("%s.xml" % host, 'w') as f:
            f.write(c)
```

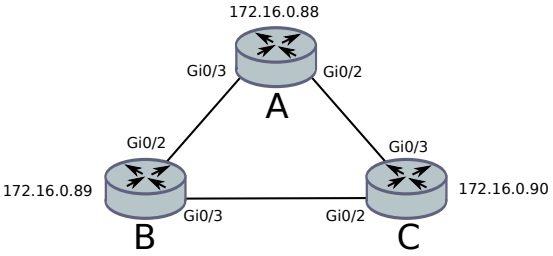
V programe na aplikovanie politík sa využíva v určitých obmenách v závislosti na type politiky. Na získanie konfigurácie zo zariadení sa používa funkcia *get_config*, na nastavenie novej konfigurácie *edit_config* a získať konfiguráciu a prevádzkové dáta je možné funkciou *get*.

Pri verifikácii konfigurácie sa na získanie prevádzkových dát používa funkcia *get*. Táto funkcia však okrem prevádzkových dát zobrazí aj aktuálnu konfiguráciu, všetko vo formáte XML. Z toho dôvodu je pred zobrazením výstupov jednotlivých príkazov potrebné spracovať tieto dáta a užívateľovi zobraziť len relevantné časti.

Kapitola 7

Realizácia

7.1 Inicializácia



Obrázek 7.1: Príklad siet'ovej topológie

Súbor, ktorý obsahuje popis topológie vo formáte JSON, sa zadá programu vo forme argumentu na príkazovom riadku (detaily sú v prílohe D. Na obrázku 7.1 je zobrazený príklad topológie a nižsie konfiguračný súbor tejto vstupnej konfigurácie.

```
{
  "devices":[
    {"hostname":"A", "ipaddr":"172.16.0.88"},
    {"hostname":"B", "ipaddr":"172.16.0.89"},
    {"hostname":"C", "ipaddr":"172.16.0.90"}
  ],
"connections":[
  [
    ["A", "GigabitEthernet0/3"],
    ["B", "GigabitEthernet0/2"]
  ],
  [
    ["A", "GigabitEthernet0/2"],
    ["C", "GigabitEthernet0/3"]
  ],
  [
    ["B", "GigabitEthernet0/3"],
    ["C", "GigabitEthernet0/2"]
```

```
  ]
]
}
```

Tento súbor obsahuje názov zariadenia (hostname), IP adresu, cez ktorú je možné vytvorit' SSH spojenie s daným zariadením, a popis spojení medzi zariadeniami. V prípade, že zariadenia existujú v simulácií prostredia VIRL a pripája sa na ne z virtuálneho počítača, na ktorom beží VIRL, používa sa zdieľaná *flat* siet', popísaná v 6.2, IP adresy by mali byt' zo siete nastavenej pre rozhranie *flat*. Program sa na základe údajov v súbore pripojí k jednotlivým zariadeniam a načíta ich konfigurácie. Tie následne uloží do tried a d'alších štruktúr, ktoré boli popísané v 5.2, aby sa dali opätovne používat'.

Okrem inicializovania programu je potrebné nastavit' základnú konfiguráciu na zariadeniach. Táto zahŕňa nastavenie SSH pripojenia a podporu protokolu NETCONF cez SSH. Ak sa využíva konfigurácia vygenerovaná nástrojom AutoNetkit, na nastavenie SSH a protokolu NETCONF sa použijú nasledujúce príkazy [26]:

```
R#configure terminal
R(config)#username cisco privilege 15 secret cisco
R(config)#ip domain-name dp.com
R(config)#crypto key generate rsa
R(config)#ip ssh version 2

R(config)#line vty 0 4
R(config-line)#transport input ssh telnet
R(config-line)#login local
R(config-line)#exit

R(config)#netconf ssh
```

Pri konfigurácii RSA kľúčov je potrebné zvolit' dĺžku najmenej 768 bitov, kvôli podpore SSH verzie 2 [26].

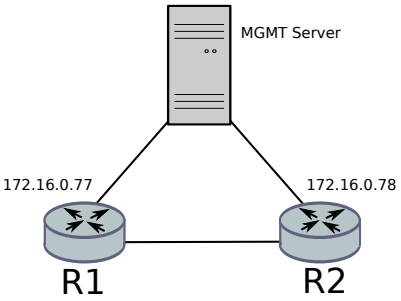
7.2 Scenár 1: nastavenie spojenia

Jedna z príležitostií, kedy pri bežnej konfigurácii môže nastat' chyba, je nastavovanie spojenia medzi dvomi zariadeniami. Je to situácia, kedy môže dôjst' k zadaniu adries z rôznych sietí alebo podsietí, môže sa stat', že užívateľ zadá nesprávne masky siete alebo zabudne zapnút' port na zariadení.

Prvý prípad (A) počíta s možnost'ou, že užívateľ poskytne nasledujúce informácie: mená zariadení, ktoré chce prepojit', čísla portov na daných zariadeniach a IP adresy s maskami, ktoré chce pre vytvárané spojenie používat'. V druhej možnosti (B) užívateľ poskytne názvy zariadení, čísla portov a IP adresu siete s maskou pre toto spojenie. Tretia možnost' (C) počíta s variantom, kedy sú zadané názvy zariadení a IP adresa siet'e s maskou. Všetky tri scenáre sú podrobnejšie popísané v nasledujúcich častiach tejto kapitoly.

7.2.1 Topológia

Na demonštráciu takého prípadu bola zvolená jednoduchá topológia pozostávajúca z dvoch smerovačov a jedného serveru, na ktorom beží konfiguračný program. Zariadenia sú prepojené tak, ako je zobrazené na obrázku 7.2. Medzi zariadeniami existuje fyzické spojenie a



Obrázek 7.2: Príklad siet'ovej topológie

zároveň sú oba smerovače pripojené k serveru. Server je v tomto prípade reprezentovaný virtuálnym počítačom prostredia VIRL. Do formátu JSON, ktorý slúži ako inicializačný vstup pre program, sa to zapíše nasledovne:

```
{
  "devices":[
    {
      "hostname":"R1",
      "ipaddr":"172.16.0.77"
    },
    {
      "hostname":"R2",
      "ipaddr":"172.16.0.78"
    }
  ],
  "connections":[
  ]
}
```

7.2.2 Konfigurácia podľa politiky

Prípad A

V tomto prípade program očakáva, že názov politiky, informácie o menách zariadení, číslach portov na týchto zariadeniach a IP adresách s maskami budú v súbore vo formáte JSON, ktorého meno sa zadá ako argument pri spustení programu. Šablóna pre túto politiku je v prílohe A.1. Potom sa zavolá metóda *connectTwoKnownPortsKnownNetwork* z modulu *ConfigureConnection*. Metóda *connectTwoKnownPortsKnownNetwork* vykoná nasledujúce kroky:

1. overí existenciu zariadení v topológii, tj. zavolá pre každé zariadenie metódu *Topology.deviceExists*
2. pomocou programu *ncclient* vytvorí SSH spojenie zo zariadeniami a získa aktuálne konfigurácie,
3. overí, či pre zadané porty neexistuje už iná konfigurácia – v prípade, že áno, vyžiada si potvrdenie od užívateľa, či má pokračovať,

4. overí, či sú porty zapnuté – ak nie, vygeneruje príkaz na ich zapnutie,
5. overí, či sa zadané IP adresy nepoužívajú niekde inde v topológií (využije uložené konfigurácie)
6. overí, či sú obe IP adresy z rovnakej siete,
7. overí, že majú zhodné siet'ové masky, nakoľko ide o priame spojenie.

Pokiaľ všetky hodnoty nespĺňajú obmedzenia politiky, vyžaduje sa od užívateľa, aby upravil hodnoty a opätovne spustil program s novou konfiguráciou. Pokiaľ vstupné hodnoty vyhovujú všetkým podmienkam, použije sa *sendConfig*, ktorá vygeneruje príkazy pre *ncclient* a zašle RCP správy protokolu NETCONF pre:

1. nastavenie IP adresy a masky na portoch
2. zapnutie portov

Prípad B

Rovnako ako v predchádzajúcom prípade, aj teraz je vstupná konfigurácia zadaná ako argument programu ako meno súboru obsahujúceho informácie vo formáte JSON, tentokrát sú to však: mená zariadení, čísla portov a IP adresa siete, ktorá sa použije pre toto spojenie. Príklad takejto konfigurácie je v prílohe A.2 Zásadným rozdielom oproti správaniu v prípade A je nutnost' vygenerovat' IP adresy, ktoré sa použijú pre vytvárané spojenie. Na tento účel sa použije metóda *getIPsFromThisNetwork* z modulu *ConfigureConnectionNetwork*, ktorá určí rozsah použiteľných IP adries. Adresy priradí portom postupne, začínajúc najnižsou možnou. Tým pádom nebude nutné vykonat' body č. 6 a č. 7 z príkladu A.

Prípad C

Ak program dostane ako vstupné hodnoty názvy zariadení a siet'ovú IP adresu a masku (príklad je uvedený v prílohe A.3), musí vygenerovat' IP adresy, ktoré použije, a taktiež určit' porty, ktoré prepojí. V tomto scenári sa predpokladá, že porty nie sú vypnuté (je na nich nastavené `no shutdown`). IP adresy, ktoré budú použité, sa určia rovnakým spôsobom ako v prípade B. Na detegovanie prepojiteľných portov sa využijú informácie protokolu CDP (Cisco Discovery Protocol [26]). Tento protokol poskytne informácie o pripojených zariadeniach, aj keď medzi nimi neexistuje nakonfigurované spojenie na L3 vrstve. Tieto informácie sa v príkazovom riadku systému IOS získavajú príkazom `show cdp neighbors`. Metóda *getCDPConfig* vygeneruje príkazy pre *ncclient*, ktorý vytvorí RPC protokolu NETCONF s týmto príkazom pre obe zariadenia. Ďalej sú dôležité informácie o existencii či neexistencii spojenia na vyššej vrstve L3. Tie je možné zobrazit' pomocou príkazu `show ip interface brief`. Pre tento príkaz sa opät' použije *ncclient*, ktorý pošle RPC správy cez NETCONF zariadeniam a zobrazí ich odpovede. Analyzovaním získaných dát určí metóda *getPortOnDev* porty, medzi ktorými je možné vytvoriť spojenie a vytvorí ho podobne ako v príklade A. Pokiaľ taká možnost' neexistuje, oznámi to užívateľovi.

7.2.3 Testovanie

Či boli vhodne nastavené IP adresy a či sa vytvorilo spojenie medzi zariadeniami je možné otestovat' viacerými možnost'ami, napr.:

- zobrazenie konfigurácie zariadení a overenie, že použité IP adresy sú nastavené a že sú nastavené pre správne porty,
- zobrazenie stavu portov pomocou príkazu `show ip interface brief`,
- spustit' príkaz `ping` medzi zariadeniami a sledovat' jeho úspešnost'.

Všetky vyššie vymenované možnosti môže manuálne spustit' na zariadeniach a overit', no takisto je možné využit' protokol NETCONF a overenie zautomatizovat'. V tejto implementácií program zobrazí ovplyvnené časti konfigurácie, v tom prípade konfiguráciu rozhraní. Ďalej zobrazí konfiguráciu rozhraní zobrazením výstupu `show ip interface brief` a overí existenciu spojenia príkazom `ping`

7.3 Scenár 2: dynamické smerovanie

Smerovanie [11] je preposielanie paketov zo zdrojovej siete do cieľovej siete, pričom sa berie do úvahy viacero faktorov, napr. ktorá cesta je najlepšia, čo sa stane, keď sa topológia zmení, ... Dynamický protokol poskytuje práve automatizovanost' a flexibilitu, čo sú kľúčové vlastnosti tejto smerovacej metódy. Pri používaní dynamického smerovacieho protokolu vedia smerovače o stave siete a upravujú smerovacie tabuľky na základe toho. Práve smerovanie je ďalším typom siet'ovej politiky, ktorý je podporovaný týmto programom. Implementovaná je určitá časť konfigurácie smerovacieho protokolu RIP a časť nastavení smerovacieho protokolu OSPF.

7.3.1 RIP

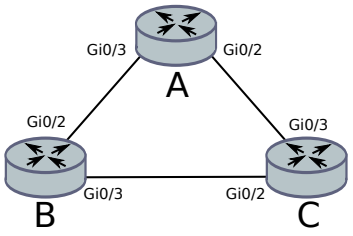
RIP [11], teda Routing Information Protocol, je smerovací protokol. Je podporovaný na rôznych platformách, vhodný pre menšie siete bez redundantných spojení a s približne rovnako rýchlymi siet'ovými linkami. Aby bolo možné používat' smerovací protokol RIP, podľa [24] je nutné

- povolit' smerovací protokol RIP,
- zasociovat' siete s protokolom RIP.

Ako sa ďalej uvádza v [24], podporované súčasti smerovacieho protokolu RIP závisia od jeho verzie, ktorá sa použije. S tým súvisí aj automatické sumarizovanie IP adries, ktoré sa dostanú do smerovacej tabuľky. Dôležitým prvkom konfigurácie smerovania je autentifikácia, ktorá vyžaduje rovnaké nastavenie na všetkých smerovačoch, a teda je náchylná na chyby počas konfigurácie. Vyššie uvedené nastavenia protokolu RIP je možné nakonfigurovat' pomocou politiky *rip*. Podobne ako pre nastavovanie IP adries na rozhraniach (7.2) existuje viacero možností ako nakonfigurovat' protokol RIP.

Topológia

Na demonštrovanie konfigurácie protokolu RIP bola vytvorená topológia zobrazená na obr. 7.3. Ide o tri smerovače zapojené za sebou. Každé spojenie medzi zariadeniami má nakonfigurované IP adresy z inej siete. Vstup na inicializáciu topológie pre konfiguračný program vyzerá nasledovne:



Obrázek 7.3: Príklad siet'ovej topológie pre RIP

```
{
  "devices": [
    {
      "hostname": "A",
      "ipaddr": "172.16.0.88"
    }, {
      "hostname": "B",
      "ipaddr": "172.16.0.89"
    }, {
      "hostname": "C",
      "ipaddr": "172.16.0.90"
    }
  ],
  "connections": [
    [
      ["A", "GigabitEthernet0/2"],
      ["B", "GigabitEthernet0/3"]
    ],
    [
      ["B", "GigabitEthernet0/2"],
      ["C", "GigabitEthernet0/3"]
    ]
  ]
}
```

Konfigurácia podľa politiky

Podobne ako pri nastavovaní IP adries (7.2), aj vstupný súbor pre konfiguráciu smerovacieho protokolu RIP obsahuje minimálne meno politiky, mená zariadení, na ktorých prebehne konfigurácia, siete, ktoré majú byt' zasociované s RIP. Ďalej tam môže byt' uvedená verzia protokolu RIP a v závislosti od nej d'alšie vlastnosti (autentifikácia a automatická sumarizácia). Zatiaľ čo meno politiky a názvy zariadení sú načítané a spracované automaticky, ostatné informácie sa spracovávajú postupne podľa ostatných nastavených hodnôt. Na základe mena politiky *rip* sa zavolá funkcia *rip* z modulu *RoutingRip* na spracovanie politiky nastavujúcej smerovanie protokolom RIP. V tejto funkcii sú spracované d'alšie parametre, či už priamo alebo pomocou volaní d'alších funkcií.

Verzia RIP

Rozhodujúcim faktorom pre riadenie toku programu je verzia protokolu RIP. Ako sa uvádza v [11], protokol RIP vo verzii 1 používa triedy adries, teda automatickú sumarizáciu a nepodporuje autentifikáciu. Určenie verzie, ktorá sa použije, je teda dôležitá informácia, ktorá je vyhodnotená ako prvá.

Podľa [11] môže nadobúdať len dve hodnoty: „1“ alebo „2“ (ďalej RIPv1 a RIPv2). Pokiaľ je vo vstupnom súbore nastavená iná hodnota, program oznámi užívateľovi, že existuje chyba v konfigurácii verzie, a skončí. Ak parameter `version` nastavený nie je vôbec, použije sa predvolené správanie systému Cisco IOS ([26]) a nastaví sa jej hodnota na „1“.

Automatická sumarizácia

Ďalším parameterom, ktorý sa spracuje, je automatická sumarizácia. Tento parameter reflektuje príkaz systému IOS `no auto-summary` [11] môže nadobúdať tiež len dve hodnoty: „on“ alebo „off“. Pokiaľ je nastavená iná hodnota, program opäť oznámi užívateľovi chybu v konfigurácii na danom mieste a skončí. Pre RIPv1 sa hodnota automatickej sumarizácie nastaví na „on“, pretože RIPv1 neumožňuje iné nastavenie automatickej sumarizácie.

Siete

Základom nastavovania smerovacieho protokolu je priradenie sietí. Podľa [11] príkaz `network x.x.x.x` jednak nastaví zverejnenie siete smerovacím protokolom a zároveň povolí zverejnenie cesty cez ľubovoľné rozhranie s adresou v rámci rozsahu danej siete. Program umožňuje nakonfigurovať siete pre smerovací protokol RIP niekoľkými spôsobmi:

- všetky siete na zariadeniach uvedených v konfiguračnom súbore politiky,
- všetky siete špecifikovné v konfiguračnom súbore,
- siete špecifikované len pre konkrétne zariadenia.

V prvom prípade je v konfigurácii politiky nastavený parameter „network“ na hodnotu „all“. Príklad tejto konfigurácie sa nachádza v prílohe A.4. Následne sa pre každé zariadenie zavolá funkcia *getAllInterfaceConfig*, ktorá zo zariadení získa konfigurácie všetkých rozhraní. V nich sú následne vyhľadané nakonfigurované IP adresy a masky. V závislosti od použitej verzie RIP sa upravia do formátu podľa tried IP adries pre RIPv1 alebo do formátu bez ohľadu na triedu IP adresy. Takto upravené adresy sú uložené v poli a neskôr je pre ne vygenerovaná RPC správa pre NETCONF.

V druhom prípade sa zoznam sietí, ktoré sa majú nastaviť v protokole RIP, uvedie priamo v konfigurácii politiky. Znamená to, že parameter „network“ obsahuje zoznam IP adries. Príklad takejto konfigurácie je uvedený v prílohe A.5. Podobne ako v predchádzajúcom prípade, aj teraz sa upravia IP adresy do správneho formátu podľa použitej verzie protokolu RIP vo funkcii *prepareNetworkIPs* a uložia sa na neskoršie spracovanie pri generovaní RPC pre NETCONF.

V poslednom prípade je pre každé ovplyvnené zariadenie uvedený zoznam sieti (príklad je zobrazený v prílohe A.6). Toto nastavenie je indikované neprítomnosťou parametru „network“ v konfigurácii politiky. Rovnako ako v predchádzajúcom prípade sa funkciou *prepareNetworkIPs* upravia IP adresy do správneho formátu podľa verzie protokolu a sú uložené pre neskoršie vygenerovanie správy pre NETCONF.

Autentifikácia

Pre protokol RIP sa autentifikácia nastavuje pomocou kľúčenky a kľúča [24]. Hodnota kľúča a autentifikačný mód musia byť rovnaké pre všetky zariadenia [24], preto sa v konfigurácii politiky (príklad je uvedený v prílohe A.7) nastavuje spoločne pre všetky zariadenia. Pokiaľ tieto dáta v konfigurácii politiky existujú, určí sa ďalší postup.

Podľa [24] RIPv1 nepodporuje smerovanie. To znamená, že pokiaľ nebude v konfigurácii politiky nastavená verzia smerovacieho protokolu na hodnotu „2“, program skončí po oznámení chyby v konfigurácii politiky. V opačnom prípade sa overí, či v konfigurácii existujú všetky informácie potrebné na nastavenie autentifikácie. Ide o nasledujúce parametre:

- keychainname – meno kľúčenky,
- key – číslo kľúča,
- keystring – kľúč,
- mode – mód (text alebo md5), nepovinný,
- privilege – úroveň prístupu (0 – 7), nepovinný.

Ako prvé sa načítajú dáta súvisiace s autentifikáciou pomocou funkcie *getAuthenticationData* z modulu *KeyChainManagement*. Táto funkcia zároveň skontroluje, či sú hodnoty validné, teda či číslo kľúča spadá do povoleného rozsahu hodnôt (0 – 65535, vrátane) alebo či hodnota parametru „mode“, ak je nastavený, je buď „text“ alebo „md5“. Pokiaľ sú nastavené všetky povinné paramatre potrebné na nastavenie autentifikácie, vráti autentifikačné dáta funkcii *rip*, kde je z nich vygenerovaná RPC správa pre NETCONF na nastavenie autentifikácie. Pred aplikovaním sa však overí, či v konfigurácii zariadení neexistuje kľúčenka a kľúč s rovnakým názvom ako novo nastavované hodnoty. Je to bezpečnostné overenie, aby nedošlo k prepísaniu kľúču bez vedomia užívateľa.

Autentifikácia sa nastavuje pre rozhrania zúčasnené v smerovaní protokolom RIP. V tejto implementácii sa pomocou funkcie *getListofAffectedInterfaces* z *Topology* získa zoznam rozhraní, na ktorých sa nastaví autentifikácia.

Konfigurácia zariadení

Po spracovaní všetkých vstupných hodnôt sa pomocou funkcie *prepareCommandRip* vytvorí časť správy RCP pre protokol NETCONF. Táto časť je následne v programe *ncclient* obalená informáciami, ktoré sú potrebné pre NETCONF, a odoslaná zariadeniu. Príklad časti takejto správy generovanej konfiguračným programom je zobrazený nižšie:

```
<config>
  <cli-config-data>
    <cmd>router rip</cmd>
    <cmd>version 1</cmd>
    <cmd>network 10.0.0.0</cmd>
  </cli-config-data>
</config>
```

7.3.2 Verifikácia

Podľa [16] okrem zobrazenia aktuálne konfigurácie zariadenia, existuje viacero príkazov na overenie nastavenia smerovacieho protokolu RIP. Medzi inými aj:

- `show ip protokols`, ktorý zobrazí smerovacie protokoly,
- `show ip route`, ktorý zobrazí smerovaciu tabuľku zariadenia.

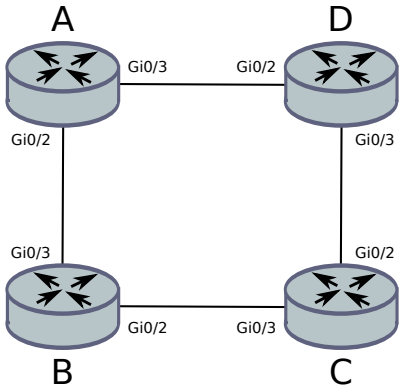
Oba tieto príkazy sú zaslané zariadeniu protokolom NETCONF ako príkazy cez program *ncclient*. Zariadenie vráti ako odpoveď informácie získané týmito príkazmi a zobrazí ich užívateľovi.

Okrem príkazov, ktoré zobrazia smerovacie informácie, je možné na overenie konektivity použiť príkaz `ping`.

7.3.3 OSPF

OSPF alebo Open Shortest Path First [11] je smerovací protokol pre väčšie siete. Ide o protokol podporujúci adresovanie, a teda aj smerovanie bez ohľadu na triedu IP adresy. Medzi jeho ďalšie vlastnosti patrí škálovateľnosť, rýchla konvergencia, nie je náchylný k smerovacím slučkám a umožňuje autentifikáciu. Na druhú stranu medzi jeho neýhody patrí v niektorých prípadoch komplexná konfigurácia. V tejto práci budú teda implementované len niektoré príkazy v sieti s jednou areou.

Topológia



Obrázek 7.4: Príklad sieťovej topológie pre OSPF

Topológia pre demonštrovanie smerovacieho protokolu OSPF je zobrazená na obr. 7.4. Formát vstupného súboru na nakonfigurovanie topológie v programe je v prílohe B.1.

Konfigurácia podľa politiky

V prípade konfigurovania protokolu OSPF, vo vstupnom súbore nesmú chýbať nasledujúce dáta:

- meno politiky,
- číslo procesu OSPF,

- číslo arey,
- názvy zariadení,
- v určitej forme siete, ktoré sa budú konfigurovat'.

Na základe mena politiky sa zavolá funkcia *ospf* z modulu *RoutingOSPF*, obsluhujúca nastavovanie parametrov pre OSPF protokol. V nej sú postupne spracovávané d'alšie parametre, na základe ktorých sa určí d'alšie správanie programu. Príklady rôznych vstupných konfigurácií sú v prílohe A.8

Číslo procesu

Ako sa uvádza v [11], v prípade OSPF sa číslo procesu (process ID) používa na povolenia viacerých inštancií protokolu OSPF na smerovači. Táto hodnota má význam len v rámci smerovača, na ktorom sa používa, tj. môže sa líšit' na smerovačoch v sieti.

Pre potreby tejto práce sa pri konfigurácii používa rovnaké ID procesu. Ide o hodnotu nastavenú v parametri „processID“, ktorá musí spadat' do rozsahu 1 – 65535, vrátane. Pokial' je v konfigurácii politiky nastavená hodnota mimo tohoto rozsahu, program oznámi užívateľovi chybu a skončí. Podobne sa správa v prípade, ak v konfigurácii táto hodnota neexistuje.

Číslo arey

Area je logický súbor sietí OSPF, smerovačov a spojení, ktoré zdieľajú určitý identifikačný atribút. Tento atribút sa nazýva *area ID* [11]. V sieti s jednou areou (single area network), sú všetky smerovače používajúce OSPF pripojené k chrbtovej (backbone) arei 0.

Area ID je 32-bitové číslo, ktoré môže byt' vyjadrené ako jedno číslo alebo vo bodkovanom decimálnom formáte (dotted decimal format), podobne ako IP adresa [11]. Program, ktorý je predmetom tejto práce, očakáva ako vstup číslo arey vo forme jedného čísla. V prípadnom d'alšom rozšírení bude možné pridat' podmienku, ktorá spracuje vstup vo formáte „dotted decimal“.

Ked'že program predpokladá siet' len s jednou areou, vhodné použitie je nastvenie parametru „areaId“ na hodnotu „0“. Pokial' sa program použije na vytvorenie súčasti konfigurácie väčšej siete s iným číslom arey, nie je zaručené vytvorenie kontektivity medzi zariadeniami.

Autentifikácia

Protokol OSPF podporuje autentifikáciu [23]. Je to časť konfigurácie, ktorá musí byt' zhodná pre všetky zariadenia, ktoré majú medzi sebou nadviazat' susedské vzt'ahy [23]. Na rozdiel od smerovacieho protokolu RIP, autentifikácia pre OSPF sa konfiguruje bez použitia kľúčenky a konfigurácia pozostáva z dvoch častí: nastavenie autentifikácie v rámci procesu OSPF a nastavenie autentifikácie pre rozhranie. Takisto existujú dva módy: textová alebo md5. Md5 je považovaná za najlepšie existujúce zabezpečenie protokolu OSPF [23].

Vo vstupnej konfigurácii politiky sa môže, ale nemusí nachádzat' časť popisujúca autentifikáciu:

```
"authentication": [
  {"mode": "text"},
  {"key": "cisco"}
]
```

Tým, že sa hodnoty parametrov „mode“ a „text“ zadávajú len na jednom mieste, a to v konfigurácii politiky, vylúči sa možnosť, že by na niektorom zo zadaných zariadení mohli byť nastavené iné hodnoty ako na ostatných.

Autentifikačné parametre sa získajú zo vstupnej konfigurácie funkciou *getAuthenticationData*. Pri ich spracovaní sa overí, či je nastavený správny mód (text alebl md5) a ak nie, užívateľovi sa oznámi chyba a program skončí. V ďalšom kroku sa získa zo vstupných dát kľúč, ktorý je spoločne s módom pripravený na ďalšie spracovanie.

Siete

Podobne ako pre RIP (7.3.1), aj pre OSPF je možné povoliť OSPF pre rôzne siete viacerými spôsobmi:

- všetky existujúce siete v použitej topológii,
- všetky siete uvedené v zozname v konfiguračnom súbore,
- siete uvedené pre jednotlivé zariadenia,
- povoliť OSPF na jednotlivých rozhraniach.

Pri konfigurovaní OSPF pre všetky existujúce siete sa zo zariadení získa konfigurácia rozhraní a z nej IP adresy, ktoré sa pripravia do správneho formátu pre OSPF (IP adresa siete a „wildcard“). Pokiaľ sú siete zadané v konfiguračnom súbore, použijú sa hodnoty odtiaľ pre všetky zariadenia. Ak v konfiguračnom súbore politiky neexistuje parameter „networks“, zoznam požadovaných sietí sa hľadá pre jednotlivé zariadenia. V prípade, že ani jedna možnosť z vyššie uvedených nie je nakonfigurovaná, program požiada užívateľa, aby upravil nastavenia, a skončí.

Konfigurácia na zariadeniach

Načítané a spracované vstupné parametre sa použijú pri vytváraní RPC správy protokolu NETCONF, ktorou sa nastavia dané hodnoty na zariadeniach. Pri vytváraní tejto správy sa zohľadňuje, ako boli nastavené siete (či globálne pre zariadenia alebo či pre jednotlivé rozhrania na nich) a či je nakonfigurovaná autentifikácia. Následne sa tieto správy pošlú pomocou programu *ncclient* na zariadenia.

Príklad takejto správy pre nastavenie sietí v procese OSPF je uvedený tu:

```
<config>
  <cli-config-data>
    <cmd>router ospf 1</cmd>
    <cmd>network 172.16.0.0 0.0.0.255 area 0</cmd>
    <cmd>network 172.16.1.0 0.0.0.255 area 0</cmd>
    </cli-config-data>
</config>
```

Verifikácia

Na overenie konfigurácie protokolu OSPF existuje množstvo rôznych príkazov [25]. Na overenie základnej funkcionality je podľa [21] možné použiť nasledujúce:

- `show ip protocol` – zobrazí smerovací protokol a siete nastavené pre tento protokol,

- `show ip ospf neighbor` – zobrazí nadviazané susedské vzťahy,
- `show ip ospf interface` – zobrazí konfiguráciu OSPF na rozhraniach,
- `show ip route` – zobrazí cesty v smerovacej tabuľke zariadenia.

Pomocou protokolu NETCONF a programu *ncclient* sa tieto príkazy spustia na zariadeniach a užívateľovi sa zobrazí ich výstup, podľa ktorého môže overiť, či boli nastavené správne hodnoty. Vstup pre *ncclient* vyzerá takto:

```
<oper-data-format-text-block>
  <exec>show ip protocol</exec>
  <exec>show ip ospf neighbor</exec>,
  <exec>show ip ospf interface</exec>
  <exec>show ip route</exec>
</oper-data-format-text-block>
```

Element `oper-data-format-text-block` určuje, že sa budú požadovať prevádzkové dáta [2].

7.4 Scenár 3: VPN

Podľa [22] a [18], konfigurácia VPN[1] pripojenia je relatívne komplexný proces pozostávajúci z viacerých krokov. Automatické vygenerovanie príkazov na základe vstupných dát môže tento proces zjednodušiť.

Vytváranie zabezpečeného IPSec VPN tunelu pozostáva z dvoch fáz: ISAKMP a IPSec (IP Security protocol [9]) [18]. Počas prvej z nich protokol ISAKMP (Internet Security Association and Key Management Protocol [13]), niekedy nazývaný aj IKE (Internet Key Exchange) vytvorí zabezpečený tunel, ktorým sa vyjedná, ako sa medzi smerovačmi vytvorí bezpečnostná asociácia (security association) pre IPSec. Na nakonfigurovanie tejto časti je nutné nastaviť nasledujúce [18]:

- politiku pre ISAKMP,
- autentifikáciu voči druhému smerovaču pre ISAKMP.

Počas druhej fázy sa vytvorí samotný IPSec tunel, ktorý šifruje dáta. Konfigurácia tohoto tunelu pozostáva z týchto častí [18]:

- vytvorenie rozšíreného ACL (Access Control List – zoznam s podmienkami, podľa ktorých sa riadi prístup [26]),
- vytvorenie transform setu[2],
- vytvorenie krypto-mapy (crypto map),
- aplikovanie krypto-mapy na verejné rozhranie.

[1] Virtual Private Network – zabezpečené pripojenie medzi dvomi sieťami [22]

[2] Transfom set je kombinácia samostatných IPSec transformácii, ktorá rozhoduje o špecifických bezpečnostných politikách pre prenos dát [12].

7.4.1 Topológia

Na reprezentovanie siete, v ktorej by bolo možné využiť VPN pripojenie, je možné použiť dva smerovače, medzi ktorými bude vytvorený tunel a za ktorými sa nachádzajú siete, ktoré majú medzi sebou komunikovať zabezpečene. Táto topológia je zobrazená na obrázku 7.5. Pre tento konfiguračný prípad sa neuvažuje, že medzi smerovačmi je nastavené prekladanie sieťových adries (NAT) z privátnych sietí. Pokiaľ by však nastavené bolo, podľa [18] by ho bolo nutné zakázať pre adresy zo siete, ktorá ma komunikovať cez zabezpečený tunel.

Obrázek 7.5: Príklad sieťovej topológie pre VPN tunel

7.4.2 Konfigurácia podla politiky

Súbor obsahujúci vstupnú konfiguráciu pre nastavenie bezpečného pripojenia vyzerá takto:

```
{
  "name": "vpn",
  "devices": {
    "A": {
      "interface": "GigabitEthernet0/1",
      "network": "10.10.10.0/24"
    },
    "B": {
      "interface": "GigabitEthernet0/1",
      "network": "20.20.20.0/24"
    }
  },
  "tunnel-mode": "IPSec",
  "IPSec": {
    "policy": {
      "priority": "2",
      "encryption": "3des",
      "hash": "md5",
      "authentication": {
        "mode": "pre-share",
        "key": "mykey"
      },
      "diffieHellmanGroup": "2",
      "lifetime": "84600"
    },
    "accessListName": "TUNN",
    "transportSetName": "VPNTS",
    "cryptoMapName": "VPNCMAP",
```

```
    "cryptoMapSeq" : "4"
  }
}
```

Okrem parametrov „accessListName“, „transportSetName“ a „cryptoMapName“ sú všetky parametre povinné a ich absencia v konfigurácii alebo nesprávne nastavenie neumožní pokračovať ďalej.

Prvým z nich je názov politiky. Po jej spracovaní sa zavolá funkcia *vpn*, v ktorej sa spracujú ostatné parametre. Všetky sú v rámci nej uložené a použijú sa ďalej pri generovaní správ pre NETCONF.

Zariadenia

Konfigurácia tejto časti obsahuje informácie o zariadeniach, medzi ktorými sa má vytvoriť zabezpečený tunel. Ďalej obsahuje názvy vonkajších rozhraní na zariadeniach, ktoré budú vytvoreným tunelom prepojené. Ich IP adresy, ktoré sú dôležité v ďalšej časti generovania konfigurácie VPN tunelu, sa získajú z aktuálnych konfigurácií zariadení.

Okrem názvov zariadení a rozhraní sú v tejto časti konfigurácie uvedené IP adresy sietí, ktoré majú spolu komunikovať cez zabezpečené pripojenie. Pred tým, než sa použijú v ďalšej konfigurácii, program overí, či sú validné.

Mód tunelu

Zabezpečený tunel môže byť nakonfigurovaný ako GRE tunel alebo ako IPSec tunel [22]. Od toho sa odvíja ďalšia časť konfigurácie. V tejto implementácii je podporovaný len mód IPSec, ale kvôli jednoduchšej príp. rozšíriteľnosti bol tento paramter zaradený do vstupnej konfigurácie politiky.

Program načíta z konfigurácie hodnotu tohoto parametru a overí, či je nastavená správne. Pokiaľ nie, oznámi užívateľovi, že konfigurácia v tomto mieste je chybná, a skončí.

IPSec: Politika

Táto časť konfigurácie obsahuje informácie potrebné pre fázu 1, teda na vytvorenie bezpečného spojenia na vyjednanie podmienok vytvárania IPSec tunelu [18].

Prvým z nich je priorita, je to číslo z rozsahu 1 až 10000, ktoré udáva prioritu danej politiky, pričom 1 znamená najvyššiu prioritu. Táto hodnota je dôležitá, ak na zariadení existuje viac zabezpečených tunelov, vyberie sa prvá, ktorá vyhovuje [18].

Ďalším je typ šifrovania. Tento parameter môže nadobúdať tri hodnoty: des, 3des alebo aes [22]. Program načíta hodnotu tohoto parametru a overí, či je to jedna z troch uvedených vyššie. Pokiaľ je zadaná iná hodnota, oznámi chybu a skončí.

Podobne sa program správa aj pre parameter „hash“, pre ktorý v tejto implementácii očakáva hodnoty „sha“ alebo „md5“.

V časti „autentication“ je uložené nastavenie módu autentifikácie, ktorý sa použije. V súčasnosti implementácia podporuje len hodnotu „pre-share“, teda zdieľaný kľúč. Hodnota kľúča sa tiež nastaví v tejto časti. Hoci nejde o parameter nutný pre nastavenie ISAKMP politiky, v tomto mieste sa nastavuje z dôvodu lepšej prehľadnosti.

Parameter „diffieHellmanGroup“ nastavuje hodnotu Diffie-Hellman grupy potrebnej pre šifrovacie algoritmy [10].

Posledným parametrom v tejto časti je „lifetime“, teda doba platnosti politiky v sekundách [22]

Ostatné parametre

Z tejto skupiny parametrov je povinný len „cryptoMapSeq“, čo je sekvenčné číslo krypto-mapy, ktoré sa použije pri jej vytváraní.

Parametre „accessListName“, „transportSetName“ a „cryptoMapName“ sú nepovinné a ich účel je umožniť pomenovať ACL, transport set, resp. krypto-mapu. Pokiaľ sa nepoužijú, alebo v nich nebudú nastavené žiadne hodnoty, použijú sa prednastavené, postupne

Konfigurácia na zariadeniach

Pre každý bod konfigurácie z fázy 1 alebo z fázy 2, ktoré sú popísané na začiatku tejto podkapitoly (7.4), je potrebné vygenerovať samostatnú RPC správu, ktorou sa nastaví požadovaná konfigurácia na zariadení pomocou nástroja *ncclient*. Týmto nástrojom sa potom nastaví nová konfigurácia najprv pre fázu 1, a potom pre fázu 2. Príklady týchto správ sú uvedené v prílohe C.

7.4.3 Verifikácia

Podľa [22] existuje množina príkazov, ktoré sa používajú na verifikáciu konfigurácie VPN tunelu. V tomto prípade zobrazí konfiguračný program výstupy z týchto príkazov dvakrát.

Prvýkrát pomocou programu *ncclient* požiada zariadenia o zobrazenie konfigurácie po fáze 1. V tomto prípade pôjde o výstup príkazu `show crypto isakmp policy`, ktorým zobrazí nastavenia politiky pre ISAMKP. Časť RPC správy pre NETCONF s týmto príkazom bude vyzerať nasledovne:

```
<oper-data-format-text-block>
  <exec>show crypto isakmp policy</exec>
</oper-data-format-text-block>
```

Druhý raz sa nastavená konfigurácia zobrazí po ukončení fázy 2. V tomto bode sa overí nastavenie ACL, krypto-mapy a transtport setu spolu s nastavením tunelu. Na tento účel sa použijú nasledujúce príkazy:

- `show access-list <name>`
- `show crypto map`
- `show crypto ipsec transform-set`

Rovnako ako predchádzajúci príkaz, aj tieto sa pošlú na zariadenia pomocou programu *ncclient* a odpoveď od zariadenia sa zobrazí ako výstup programu.

Kapitola 8

Možné rozšírenia

Konfiguračný program, ktorý je predmetom tejto práce, pokrýva len určité aspekty konfigurácie siet'ových zariadení. Nastavenie pripojení, smerovania, prístupu a d'alších siet'ových služieb je komplexná problematika.

Z hl'adiska väčšieho užívateľského komfortu by bolo možné v rámci rozšírení implementovat' grafické užívateľské rozhranie. Táto možnost' by v určitých aspektoch zjednodušila konfiguráciu. Užívateľ by priamo videl, aké možnosti má k dispozícii. Zároveň by bolo možné dynamicky kontrolovat' ich kombináciu s ostatnými zvolenými hodnotami, príp. zobrazovat' len možnosti, ktoré nie sú v konflikte s už vybranými.

Spravovanie konfigurácie a možnost' vrátit' zmeny naspät' by mohla byt' uskutočnieľná zmena. Už v súčasnej verzii poskytuje program ukladanie konfigurácií pred ich zmenou, ale užívateľ ich v prípade potreby musí nahrat' manuálne.

Ďalším možným rozšírením by mohlo byt' doplnenie vlastností smerovacích protokolov. Mohlo by íst' o d'alšie parametre pre už existujúci RIP alebo OSPF, alebo o pridanie d'alších smerovacích protokolov, alebo o implementáciu redistribúcie smerovacích informácií medzi rôznymi protokolmi.

Tunely a zabezpečené pripojenie poskytujú d'alšie možnosti, ktoré sa v súčasnej implementácii nenachádzajú. Bolo by možné doplnit' generovanie konfigurácie napr. pre GRE tunel.

Taktiež by bolo možné doplnit' úplne nové moduly, či už určitú formu implementácie prístupových zoznamov (ACL), dohôd o kvalite služieb (SLA – Service Level Agreement) alebo multimediálne služby, ako napr. VoIP (Voice over IP, hlasové služby).

Praktickým rozšírením by mohlo byt' takisto odtienenie rozdielnych formátov konfiguračných príkazov pre rôzne architektúry, čo by uľahčilo konfiguráciu sietí so zariadeniami od rôznych výrobcov.

Súčasný dizajn programu používa triedy a moduly na manipulovanie so vstupnou konfiguráciou spoločné pre všetky politiky, no politiky samotné používajú separátne moduly. Z toho hl'adika by mohlo byt' pridanie nového modulu alebo doplnenie funkcionality k už existujúcej politiky realizovateľné bez výraznejších problémov.

Kapitola 9

Záver

Predmetom tejto práce bolo zoznámiť sa s prostriedkami na konfiguráciu sieťových zariadení a s prostriedkami na popis sieťovej konfigurácie a politík, vytvoriť jazyk, ktorý je schopný tieto politiky popísať, vytvoriť model siete, nad ktorým sa tieto politiky aplikujú a implementovať generovanie a nastavenie konfigurácie na zariadeniach.

Výsledkom práce je konfiguračný program, ktorý umožňuje aplikovať určité politiky na sieť a tento text, ktorý popisuje program a súvisiace prostriedky.

Kapitola 2 sa venuje popisu existujúcich prostriedkov na popis politík a prostriedkov na ich aplikáciu. Kapitola 3 potom prezentuje spôsob popísania politík pre účely tejto práce. Sieťový model navrhnutý v kapitole 5 je graf, a hoci sa v súčasnej implementácii využíva minimálne, pre prípadne ďalšie rozšírenia je kľúčovým prvkom. V kapitole 4 je spomenutý nástroj ConfigAssure, ktorý overuje politiky a nastavenia konfigurácie formálne. V tejto práci boli na verifikáciu politík použité dva prístupy: kontrola dát pred ich aplikáciou na zariadenie a zobrazenie kontrolných informácií zo zariadení po aplikácií politík.

Druhá časť práce sa venuje príkladom generovania sieťovej konfigurácie a jej nastaveniu na zariadeniach. Boli vybraté príklady, na ktorých je možné demonštrovať zjednodušenie, ktoré poskytne automatické generovanie konfigurácie. Či už ide o nesprávne zadané sieťové masky alebo rozdielne heslo pre autentifikáciu alebo typografickú chybu pri nastavovaní VPN tunelu.

V závere práce sú popísané rozličné možné rozšírenia, ktorých je vzhľadom na povahu sieťovej konfigurácie veľké množstvo.

Literatura

[1] Al-Shaer, E.; Marrero, W.; El-Atawy, A.; aj.: *Network configuration in a box: towards end-to-end verification of network reachability and security.* In *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*, Oct 2009, ISSN 1092-1648, s. 123–132.

[2] Bhushan, S.; Poulopoulos, L.: *ncclient – Documentation* . 2014, [Online; navštívené 2.5.2016].
URL http://ncclient.readthedocs.io

[3] Bhushan, S.; Poulopoulos, L.: *ncclient.* 2016, [Online; navštívené 2.5.2016].
URL http://ncclient.org/

[4] Bhushan, S.; Poulopoulos, L.: *ncclient.* 2016, [Online; navštívené 2.5.2016].
URL https://github.com/ncclient/ncclient

[5] Bjorklund, M.: *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)* . Technická zpráva, Október 2010, [Online; navštívené 15.1.2015].
URL https://tools.ietf.org/html/rfc6020

[6] Clark, D.: *Policy Routing in Internet Protocols, RFC 1102* . Technická zpráva, Máj 1989, [Online; navštívené 8.12.2014].
URL http://tools.ietf.org/html/rfc1102

[7] Enns, R.; Bjorklund, M.; Schoenwaelder, J.; aj.: *Network Configuration Protocol (NETCONF)* . Technická zpráva, Jún 2011, [Online; navštívené 15.1.2015].
URL https://tools.ietf.org/html/rfc6241

[8] Inc, C. S.: *Cisco Networking Services Configuration Guide, Cisco IOS Release15M&T* . 2012-11-22, [Online; navštívené 15.11.015].
URL http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/cns/configuration/15-mt/cns-15-mt-book.pdf

[9] Kent, S.; Atkinson, R.: *https://tools.ietf.org/html/rfc2401, RFC 2401* . Technická zpráva, November 1998, [Online; navštívené 18.5.2015].
URL https://tools.ietf.org/html/rfc2401

[10] Kivinen, T.; Kojo, M.: *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE), RFC 3526* . Technická zpráva, Máj 2003, [Online; navštívené 18.5.2015].
URL https://tools.ietf.org/html/rfc3526

[11] Macfarlane, J.: *Network Routing Basics: Understanding IP Routing in Cisco Systems*. John Wiley & Sons, 2006, ISBN 0470038829.

[12] Mason, A.: *IPSec Overview Part Two: Modes and Transforms*. 2002, [Online; navštívené 16.5.2016].
URL http://www.ciscopress.com/articles/article.asp?p=25477&seqNum=3

[13] Maughan, D.; Schertler, M.; Schneider, M.; aj.: *Internet Security Association and Key Management Protocol, RFC 2408* . Technická zpráva, November 1998, [Online; navštívené 18.5.2015].
URL https://tools.ietf.org/html/rfc2408

[14] Narain, S.: *Network Configuration Management via Model Finding*. In *Proceedings of the 19th Conference on Large Installation System Administration Conference - Volume 19*, LISA '05, Berkeley, CA, USA: USENIX Association, 2005, s. 15–15.
URL http://dl.acm.org/citation.cfm?id=1251150.1251165

[15] Narain, S.; Levin, G.; Malik, S.; aj.: *Declarative Infrastructure Configuration Synthesis and Debugging*. *Journal of Network and Systems Management*, ročník 16, č. 3, 2008: s. 235–258, ISSN 1064-7570.
URL http://dx.doi.org/10.1007/s10922-008-9108-y

[16] Odom, W.: *OSPF Implementation*. 2002, [Online; navštívené 16.5.2016].
URL http://www.ciscopress.com/articles/article.asp?p=26421&seqNum=4

[17] Rahman, M. A.; Al-Shaer, E.: *A Formal Framework for Network Security Design Synthesis*. In *ICDCS*, IEEE, 2013, s. 560–570.
URL http://dblp.uni-trier.de/db/conf/icdcs/icdcs2013.html#RahmanA13

[18] Singh, R.: *Configuring site to site IPSec VPN tunnel between Cisco routers*. 2012, [Online; navštívené 16.5.2016].
URL http://www.firewall.cx/cisco-technical-knowledgebase/cisco-routers/867-cisco-router-site-to-site-ipsec-vpn.html

[19] Strassner, J.; Ellesson, E.: *Terminology for describing network policy and services* . Technická zpráva, Jún 1999, [Online; navštívené 15.1.2016].
URL
http://www.ietf.org/proceedings/46/I-D/draft-ietf-policy-terms-00.txt

[20] Strassner, J.; Schleimer, S.: *Policy Framework Definition Language* . Technická zpráva, November 1998, [Online; navštívené 8.12.2014].
URL https://tools.ietf.org/html/draft-ietf-policy-framework-pfdl-00

[21] Teare, D.; Graziani, R.; Vachon, B.: *OSPF Implementation*. 2015, [Online; navštívené 16.5.2016].
URL http://www.ciscopress.com/articles/article.asp?p=2294214

[22] WWW stránky: *Cisco IOS VPN Configuration Guide*. [Online; navštívené 16.5.2016].
URL http://www.cisco.com/c/en/us/td/docs/security/vpn_modules/6342/vpn_cg/6342site3.html

[23] WWW stránky: *Sample Configuration for Authentication in OSPF*. 2005, [Online; navštívené 14.5.2016].
URL http://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/13697-25.html

[24] WWW stránky: *Cisco IOS IP Configuration Guide, Release 12.2*. 2014, [Online; navštívené 5.5.2016].
URL http://www.cisco.com/c/en/us/td/docs/ios/12_2/ip/configuration/guide/fipr_c/1cfrip.html

[25] WWW stránky: *Cisco IOS IP Routing: OSPF Command Reference*. 2015, [Online; navštívené 20.5.2016].
URL http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_ospf/command/iro-cr-book/ospf-s1.html

[26] WWW stránky: *Cisco Systems, Inc.* . 2016, [Online; navštívené 16.4.2016].
URL http://www.cisco.com

[27] WWW stránky: *Juniper Networks* . 2016, [Online; navštívené 16.4.2016].
URL http://www.juniper.net

[28] WWW stránky: *Python Course* . 2016, [Online; navštívené 12.5.2016].
URL http://www.python-course.eu

[29] WWW stránky: *VIRL – Virtual Internet Routing Lab*. 2016, [Online; navštívené 20.4.2016].
URL http://virl.cisco.com/

Přílohy

Seznam příloh

Príloha A

Šablóny

A.1 Konfigurácia spojenia: prípad A

```
{
    "name": "connectTwoKnownPortsKnownNetwork",
    "devices": {
        "R1" : {
            "interface":"GigabitEthernet0/1",
            "ipaddr":"10.0.11.2/24"
        },
        "R2" : {
            "interface":"GigabitEthernet0/1",
            "ipaddr":"10.0.11.3/24"
        }
    }
}
```

A.2 Konfigurácia spojenia: prípad B

```
{
    "name": "connectTwoKnownNetwork",
    "devices": {
        "R1" : {
            "interface":"GigabitEthernet0/1",
            "network":"10.0.11.0/24"
        },
        "R2" : {
            "interface":"GigabitEthernet0/1",
            "network":"10.0.11.0/24"
        }
    }
}
```

A.3 Konfigurácia spojenia: prípad C

```
{
    "name": "connectTwo",
    "devices" : {
        "R1" : {
            "network":"10.0.11.0/24"
        },
        "R2" : {
            "network":"10.0.11.0/24"
        }
    }
}
```

A.4 Routing RIP: prípad A

```
{
    "name": "rip",
    "devices": [
        "A",
        "B",
        "C"
    ],
    "networks": "all",
    "version": "1"
}
```

A.5 Routing RIP: prípad B

```
{
    "name": "rip",
    "devices": [
        "A",
        "B",
        "C"
    ],
    "networks": [
        "10.0.0.0",
        "11.0.0.8/12",
        "12.0.8.0/24"
    ],
    "version": "1"
}
```

A.6 Routing RIP: prípad C

```
{
```

```
    "name": "rip",
    "devices": {
        "A": [
            "11.0.0.0",
            "12.0.0.0"
        ],
        "B": [
            "12.0.0.0",
            "13.0.0.0"
        ],
        "C": [
            "11.0.0.0",
            "13.0.0.0"
        ]

    },
    "version": "2",
    "auto-summary": "off"
}
```

A.7 Routing RIP: prípad D

```
{
    "name": "rip",
    "devices": [
        "A",
        "B",
        "C"
    ],
    "networks": "all",
    "version": "2",
    "autoSummary": "off",
    "key": [
        {
            "keychainname": "routingchain"
        },
        {
            "key": "2"
        },
        {
            "keystring": "router"
        },
        {
            "mode": "text"
        }
    ]
}
```

A.8 Routing OSPF: Všetky siete

```
{
    "name": "ospf",
    "processId" : "1",
    "area" : "0",
    "devices": [
        "A",
        "B",
        "C",
        "D"
    ],
    "networks":"all",
    "authentication": [
        {"mode": "text"},
        {"key": "nieco"}
    ]
}
```

A.9 Routing OSPF: Špecifikované siete

```
{
    "name": "ospf",
    "processId" : "1",
    "area" : "0",
     "devices": [
        "A",
        "B",
        "C",
        "D"
    ],
    "networks": [
        "10.0.0.0/24",
        "11.0.0.8/12",
        "12.0.8.0/24"
    ]
}
```

A.10 Routing OSPF: Siete špecifikované pre zariadenia

```
{
    "name": "ospf",
    "processId" : "1",
    "area" : "0",
    "devices": {
        "A": [
            "12.0.0.0/24",
```

```
            "13.0.0.0/24"
        ],
        "B": [
            "12.0.0.0/24",
            "11.0.0.0/12"
        ],
        "C": [
            "11.0.0.0/12",
            "13.0.0.0/24"
        ],
        "D": [
            "11.0.0.0/12"
        ]
    },
    "authentication": [
        {"mode": "text"},
        {"key": "nieco"}
    ]
}
```

A.11 Routing OSPF: OSPF povolené na rozhraniach

```
{
    "name": "ospf",
    "processId" : "1",
    "area" : "0",
    "devices": {
        "A": {
            "interfaces" : [
                "GigabitEthernet0/2",
                "GigabitEthernet0/3"
            ]
        },
        "B": {
            "interfaces" : [
                "GigabitEthernet0/2",
                "GigabitEthernet0/3"
            ]
        },
        "C": {
            "interfaces" : [
                "GigabitEthernet0/2",
                "GigabitEthernet0/3"
            ]
        },
        "D": {
            "interfaces" : [
                "GigabitEthernet0/2",
```

```
                "GigabitEthernet0/3"
            ]
        }
    },
    "authentication": [
        {"mode": "text"},
        {"key": "nieco"}
    ]
}
```

Příloha B

Topológia

B.1 OSPF

```
{
  "devices": [{
        "hostname": "A",
        "ipaddr": "172.16.0.50"
    }, {
        "hostname": "B",
        "ipaddr": "172.16.0.51"
    }, {
        "hostname": "C",
        "ipaddr": "172.16.0.52"
    }, {
        "hostname": "D",
        "ipaddr": "172.16.0.53"
    }],
    "connections": [
        [
            ["A", "GigabitEthernet0/2"],
            ["B", "GigabitEthernet0/3"]
        ],
        [
            ["B", "GigabitEthernet0/2"],
            ["C", "GigabitEthernet0/3"]
        ],
        [
            ["D", "GigabitEthernet0/3"],
            ["C", "GigabitEthernet0/2"]
        ],
        [
            ["A", "GigabitEthernet0/3"],
            ["D", "GigabitEthernet0/2"]
        ]
    ]
}
```

Příloha C

Konfiguračné správy pre VPN

C.1 Fáza 1: ISAKMP

```
<cmd>crypto isakmp policy 2</cmd>
<cmd>encryption 3des</cmd>
<cmd>hash md5</cmd>
<cmd>authentication pre-share</cmd>
<cmd>group 2</cmd>
<cmd>lifetime 86400</cmd>
```

C.2 Fáza 2: IPSec

C.2.1 ACL

```
<cmd>ip access-list extended TUNN</cmd>
   <cmd>permit ip 10.10.10.0 0.0.0.255 20.20.20.0 0.0.0.255</cmd>
```

C.2.2 Transport Set

```
<cmd>crypto ipsec transform-set VPNTS esp-3des esp-md5-hmac</cmd>
```

C.2.3 Crypto Map

```
<cmd>crypto map VPNCM 4 ipsec-isakmp</cmd>
<cmd>set peer 2.2.2.3</cmd>
<cmd>set transform-set VPNTS</cmd>
<cmd>match address TUNN</cmd>
```

C.2.4 Aplikácia crypto mapy na rozhranie

```
<cmd>interface GigabitEthernet0/1</cmd>
<cmd>crypto map VPNCM</cmd>
```

Příloha D

Readme.txt

```
README
-----------------------------------------------------------------
Prosím, prečítajte si tieto inštrukcie pre spustením programu
-----------------------------------------------------------------
Pred spustením:

prosím, nainštalujte:
    - netaddr
    - ipaddress vo verzii pre python2
    - ncclient 0.4.7

Kód je kompatibilný s Python 2.7

Na sieťových zariadeniach je nutné mať povolený prístup cez SSHv2 a nastavený
NETCONF cez SSH ((config)#netconf ssh).

Prístupové údaje k zariadeniu sa umiestňujú do súboru access.json buď pre
všetky zariadenia, alebo pre každé zariadenie zvlášť.

Program sa spúšťa z priečinku, v ktorom sa nachádza, nasledovne:

./configure_connection.py -t <topology file> -p <policy file>

Topology file je súbor obsahujúci popis topológie, policy file obsahuje popis
politiky, ktorá sa má konfigurovať. Príklady týchto súborov sú umiestnené
v priečinkoch Topologies a Policies.
```