VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ŘÍDICÍ JAZYK PRO OPENFLOW SÍTĚ

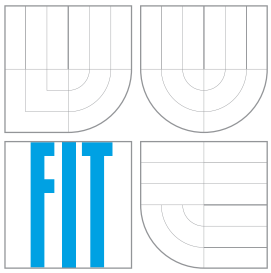
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DÁVID ANTOLÍK

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ŘÍDICÍ JAZYK PRO OPENFLOW SÍTĚ

A NETWORK CONTROL LANGUAGE FOR OPENFLOW NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

DÁVID ANTOLÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ONDŘEJ RYŠAVÝ, Ph.D.

BRNO 2013

Abstrakt

Cílem této bakalářské práce je navrhnout vysoko-úrovňový jazyk OpenFlow-Network Control Language pro popis síťové topologie a chování počítačových sítí. Překladač jazyka a jeho interpretr byl implementován pomocí Trema OpenFlow kontroléru. Výsledná implementace byla otestována ve virtuálním prostředí, kde byla demonstrována její plná funkčnost pro základní síťové konfigurace.

Abstract

The main goal of this bachelor thesis is to design and implement a high-level language for declarative configuration of Open-Flow networks. The compiler of the language, data model and interpreter were implemented employing Trema as the underlaying Open-Flow controller. The implementation was tested in a virtual network environment and it was shown that it is fully functional for basic network configurations.

Klíčová slova

počítačové sítě, softwarově definované sítě, openflow

Keywords

computer networks, software-defined networks, openflow

Citace

Dávid Antolík: Řídicí jazyk pro OpenFlow sítě, bakalářská práce, Brno, FIT VUT v Brně, 2013

Řídicí jazyk pro OpenFlow sítě

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Ondřeja Ryšavého, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Dávid Antolík
14. května 2013

Poděkování

Ďakujem vedúcemu mojej bakalárskej práce pánovi Ing. Ondřejovi Ryšavému, Ph.D. za jeho čas, hodnotné pripomienky a podnety k jej riešeniu.

Obsah

Kapitola 1

Úvod

Cieľom mojej bakalárskej práce je vytvorenie vysoko-úrovňového deklaratívneho jazyka slúžiaceho k popisu topológie a riadenia počítačovej siete na báze technológie OpenFlow. Tento jazyk môže slúžiť pre administrátorov počítačových sietí. Obsahuje prostriedky, ktoré umožňujú popísať sieťové zariadenia, prepojenia medzi nimi a požadované chovanie siete. Chovanie siete je realizované na základe politík, deklarovaných pomocou tohto jazyka.

Súčasné počítačové siete sú tvorené zo sieťových prvkov. Prevažná časť sieťových prvkov obsahuje vlastnú konfiguráciu, ktorá je do zariadenia staticky vložená administrátorom. Na základe konfiguračných pravidiel vykonáva zariadenie svoju funkciu. To môže byť preposlanie dát do rôznych segmentov siete, filtrovanie dát na rôznych vrstvách OSI modelu, zmeny v hlavičkách paketov alebo v samotných dátach. Architektúra OpenFlow upúšťa od tohto spôsobu a umožňuje definovať chovanie počítačovej siete centralizovane. V sieti je potom prítomný kontrolér, ktorý plní úlohu arbitra.

Počítačové siete tvoria základ pre komunikáciu. Sú každodenne využívané vládami, spoločnosťami aj súkromnými osobami. Slúžia k zabezpečeniu firemnej infraštruktúry, k dorozumievaniu sa na veľké vzdialenosti a k prenosu informácií. Umožňujú organizovanie, obchodovanie, plánovanie postupov a ďalšie aktivity. Súčasný obrovský rozsah možností využívania počítačových sietí vyžaduje ich maximálnu spoľahlivosť. K spoľahlivosti sa pripájajú ďalšie požiadavky. Dôležitá je šírka prenosového pásma s čím úzko súvisí rýchlosť komunikácie a odozvy siete. Veľmi dôležitá je bezpečnosť počítačových sietí a ochrana pred útokmi. Udržať správne fungovanie spomenutých faktorov je náročné z dôvodu veľkého rozsahu sietí. S veľkým počtom zariadení v sieťovom segmente výrazne rastú nároky na správu a údržbu siete. Preto pri výskyte problému v sieti môže byť náročné rýchle nájdenie príčiny a jej odstránenie, čo môže zhoršovať aj reťazové šírenie chyby sieťou.

Z uvedených dôvodov je vhodné navrhnúť a realizovať riešenia, ktoré umožnia automatizované a centrálne riadenie počítačových sietí na základe popisu ich chovania. Chovanie konkrétneho sieťového zariadenia teda neurčuje jeho konfiguračný súbor, ale arbiter. Ten na základe popisu chovania siete nájde optimálne riešenia pre zabezpečenie požadovanej funkcionality. To so sebou prináša veľkú výhodu, keďže pre riešenie jedného problému je niekedy k dispozícii niekoľko správnych postupov. V prípade, že jeden z nich postupom času zlyhá a nedokáže zabezpečiť požadovanú funkciu, aplikuje sa iný. Takáto situácia môže v sieti nastať napríklad pri výpadku niektorého zariadenia alebo nedostupnosti linky.

Jazyk navrhnutý v mojej práci bude jednoduchý, zrozumiteľný a pre administrátora pohodlný. Poskytne mu prostriedky pre popis chovania sieťovej topológie. Jednoduchým zápisom v tomto programovacom jazyku bude administrátor schopný vytvoriť riešenie konkrétnej situácie v počítačovej sieti.

1.1 Softvérovo definované siete

Záujem odbornej verejnosti o architektúru softvérovo definovaných sietí v súčasnosti prudko rastie. Základnou myšlienkou týchto sietí je oddelenie riadiacej vrstvy od prenosovej vrstvy sieťových zariadení. Do siete to prináša nové zariadenie, ktoré preberá úlohu riadenia. Toto zariadenie sa často nazýva kontrolér. Kontrolér je pripojený k sieťovému zariadeniu a komunikuje s ním pomocou špeciálenho protokolu (napríklad OpenFlow). Počítačové siete sa obvykle skladajú z veľkého počtu sieťových prvkov. Kontrolér musí byť schopný obslúžiť všetky tieto zariadenia. Kvôli zvýšeniu výkonnosti a spoľahlivosti SDN sietí je možné priniesť do siete aj viac kontrolérov. V prípade, že jeden z nich zlyhá, prevezme funkciu arbitra iný kontrolér. Súčasné zapojenie viacerých kontrolérov môže pomôcť aj pri zvýšovaní výkonnosti siete. Veľkou výhodou softvérovo definovaných sietí je možnosť využitia nenáročných sieťových zariadení. Akákoľvek rozhodovacia a riadiaca činnosť je z týchto zariadení presunutá na arbitra. Takéto sieťové zariadenia môžu byť lacnejšie a menej náchylné na chyby. Hlavnou jednotkou, s ktorou sa v softvérovo definovaných sieťach pracuje, je sieťový tok.

1.1.1 Sieťový tok

Sieťový tok je možné charakterizovať ako postupnosť paketov so spoločnou vlastnosťou. Tieto pakety prechádzajú bodom pozorovania, ktorým môže byť napríklad sieťové zariadenie, prípadne konkrétny port tohto zariadenia. Dôležitou veličinou pri sledovaní sieťových tokov je čas. Sieťový tok možno sledovať v určitom, pevne stanovenom časovom intervale. K vlastnostiam, podľa ktorých je možné rozlišovať sieťové toky patrí napríklad:

- Zdrojová a cieľová IP adresa paketu
- Zdrojová a cieľová MAC adresa paketu
- Číslo fyzického portu switcha, na ktorom bol paket prijatý
- Typ protokolu L3 a L4
- Iné parametre hlavičky paketu

1.1.2 Klasifikácia sieťových tokov

Založená je na klasifikácii jednotlivých paketov. Rozbalí sa PDU[1] až na príslušnú vrstvu, na ktorej sú extrahované potrebné údaje. Na základe týchto údajov môže byť paket zaradený do príslušného sieťového toku.

[1] PDU - protocol data unit. Označuje dátovu jednotku na príslušnej vrstve OSI modelu.

Kapitola 2

Protokol a architektúra OpenFlow

Architektúra OpenFlow [6] sa niekedy nesprávne označuje za synonymum softvérovo definovaných sietí. V skutočnosti je OpenFlow iba nástrojom, ktorý túto myšlienku uplatňuje. OpenFlow je protokol, ktorý vytvára komunikačný kanál medzi OpenFlow switchom a OpenFlow kontrolérom.

Odkedy sa hovorí o softvérovo definovaných sieťach, diskutuje sa aj problematika ich výkonnosti. Výkonnosť v tomto prípade spočíva v priepustnosti siete a rýchlosti odozvy. Problém môže nastať v sieťach, ktoré spracovávajú veľký počet relatívne malých dátových tokov. Keďže OpenFlow switch nemá prostriedky na to, aby rozhodol o smerovaní neznámeho dátového toku, ktorý nie je obsiahnutý vo Flow-table, rozhodnutie vykoná kontrolér.

Kontrolér musí bezodkladne analyzovať paket a zistiť, aká akcia naň bude aplikovaná. Následne oznámi switchu nové pravidlo. Toto pravidlo hovorí, ako má switch ďalej zaobchádzať s paketmi z tohto dátového toku. Pravidlo obsahuje popis paketov z dátového toku, tzv. `Match` a akcie, ktoré budú na tento dátový tok aplikované.

2.1 Prvky architektúry OpenFlow

2.1.1 OpenFlow switch

OpenFlow switch [7] je zariadenie, ktoré aplikuje príslušné pravidlá z internej Flow-Table na dáta ním prechádzajúce. Pakety, ktoré sú prijaté na rozhraní switcha sa pokúsi zaradiť do správneho sieťového toku. Ak zodpovedá niektorému sieťovému toku vo Flow-table (kapitola 2.1.2), aplikuje sa naň zodpovedajúca akcia. Pokiaľ switch nie je schopný priradiť paket k žiadnemu existujúcemu sieťovému toku, je tento paket preposlaný arbitrovi. Switch sa takto dotazuje arbitra, aby zistil, ako má s týmto paketom zaobchádzať. Arbiter na základe hlavičiek, prípadne významu dát v tele paketu určí, ktorá akcia sa má aplikovať.

Switch potom na základe Flow-Table umožňuje smerovať sieťové toky do dostupných segmentov siete, riadiť a kontrolovať prístupy na základe pôvodu paketov alebo prerozdeľovať záťaž medzi dostupné zariadenia. Všetky tieto operácie vykonáva výlučne na základe pravidiel v jeho internej Flow-table, ktoré tam nastavil arbiter.

Základné akcie, ktoré podporuje OpenFlow switch sú:

- Preposlanie paketu cez jeden alebo viac výstupných portov zariadenia.
- Enkapsulácia paketu a odoslanie na kontrolér, ktorý paket analyzuje a na základe jeho obsahu rozhodne o akcii, ktorú vykoná. Podľa tohto paketu môže zároveň špecifikovať nový sieťový tok a akciu, ktorá sa na tento sieťový tok bude aplikovať.

- Paket, pre ktorý neexistuje akcia (nie je zatial stanovená) a je v sieti nežiaduci, sa zahodí. Zahodenie paketu, môže byť použité z dôvodu bezpečnosti, obmedzenia útoku DoS prípadne na redukovanie počtu zbytočných broadcastových správ, ktoré prechádzajú sieťou.

Niektoré switche podporujú okrem architektúry OpenFlow aj funkciu bežného switcha. Vtedy na takto klasifikované pakety aplikujú štvrtú akciu:

- Preposlanie paketu cez switch zvyčajným spôsobom, ako u klasického L2 switcha. Obsahuje teda tabuľku mapovania MAC adries na čísla portov.

Takáto kombinácia bežného switcha s podporou OpenFlow architektúry sa používa v prípade, ak je žiadúce experimentovať s existujúcou sieťou, pričom predchádzajúca funkčnosť tejto siete musí zostať zachovaná. Ďalšou možnosťou, ako oddeliť experimentálne prostredie od produkčného, je použitie VLAN sietí.

OpenFlow switch pozostáva z komponent, ktoré zabezpečujú jeho funkciu. Sú to minimálne 3 základné časti:

Flow-table
: Obsahuje špecifikáciu a akcie definované pre sieťové toky.

OpenFlow komunikačný kanál
: Zabezpečený kanál medzi arbitrom (v OpenFlow sieťach napĺňa Flow-table a rozhoduje o akciách, ktoré budú vykonané s jednotlivými paketmi, nazýva sa tiež kontrolér) a switchom.

OpenFlow protokol
: Definuje štandard pre komunikáciu medzi kontrolérom a switchom, obsahuje prostriedky pre správu pravidiel vo Flow-table.

2.1.2 Tabuľka tokov (Flow-table)

Tabuľka tokov má v OpenFlow sieťach významnú úlohu, najmä v OpenFlow switchoch. Obsahuje údaje pre klasifikáciu jednotlivých PDU, na základe ktorých sa určí ich príslušnosť k dátovému toku. Tok je možné symbolicky popísať ako jeden riadok v tejto tabuľke. Ďalej ku každému dátovému toku definuje súbor akcií, ktoré sa majú vykonať pri doručení paketu, patriaceho do tohto toku. Flow-table teda môže v hlavičke obsahovať:

- Identifikátor dátového toku
- Hlavičku dát pre určenie príslušnosti k dátovému toku
- Súbor akcií, ktoré sa aplikujú na dátový tok
  - preposlať paket cez niektoré fyzické rozhranie
  - zmeniť údaje v hlavičke paketu (napríklad MAC adresu, IP adresu)
  - zahodiť paket
- Dĺžku životnosti dátového toku, ktorá môže byť
  - pevne stanovená v časových jednotkách (tzv. hard-timeout)
  - počet časových jednotiek od prijatia posledného paketu, po ktorých bude sieťový tok ešte existovať (tzv. idle-timeout)

2.1.3 OpenFlow kontrolér

OpenFlow kontrolér plní v OpenFlow sieťach funkciu arbitra. Je to softvérová aplikácia. S OpenFlow zariadeniami komunikuje prostredníctvom OpenFlow protokolu cez zabezpečený kanál. Kontrolér je prostriedok, ktorý riadi softvérovo definované siete. Má prehľad o stave sieťovej topológie a môže súčasne zasahovať do rozličných sieťových segmentov. V súčasnosti už existuje množstvo softvérových OpenFlow kontrolérov. Väčšinu z nich je možné spustiť na ľubovolnej architektúre. Vyhovujúca môže byť aj bežná pracovná stanica. Pre kontroléry v náročnejších sieťach je samozrejme nutné vyhradiť väčší výpočtový výkon, keďže OpenFlow sieť je závislá práve na riadení, ktoré kontrolér vykonáva.

2.2 Prehľad dostupných OpenFlow kontrolérov

Pri výbere vhodného OpenFlow kontroléra je možné orientovať sa podľa niekoľkých kritérií. V závislosti od typu počítačovej siete, v ktorej bude kontrolér použitý, sa môže klásť dôraz na jeho výkonnosť a rýchlosť spracovania požiadaviek. Ďalšiu variantu predstavuje použitie kontroléra, ktorý umožní rýchly vývoj prototypu a nasadenie v sieti. Takéto kontroléry sú flexibilné a vykonanie zmien prípadne zakomponovanie nových funkcií je väčšinou bezproblémové.

FloodLight
: FloodLight je zaujímavým kontrolérom pre softvérovo definované siete, ktoré využívajú protokol OpenFlow. Je uvoľnený pod licenciou Apache a umožňuje skutočne široké využitie. FloodLight je vyvíjaný otvorenou komunitou s účasťou vývojarov z BigSwitch Networks [1]. Je pomerne kvalitne zdokumentovaný a testovaný. Tento kontrolér je napísaný v jazyku Java. Ponúka modulárny systém, ktorý umožňuje ľahké rozširovanie. Navrhnutý je pre dosahovanie vysokej výkonnosti v OpenFlow sieťach. Inštalácia kontroléra FloodLight je jednoduchá a zaberie minimum času.

NOX a príbuzné kontroléry
: Je tradičný a zrejme najrobustnejší OpenFlow kontrolér, ktorý poskytuje API v jazyku C++. Podporuje OpenFlow protokol verzie 1.0. Alternatíva k tomuto kontroléru je POX. Rozdiel je v tom, že využíva jazyk Python. Jaxon je OpenFlow kontrolér založený na Jave, vytvára tenkú vrstvú nad kontrolérom NOX a je na ňom závislý.

Beacon
: Beacon je rýchly a modulárny OpenFlow kontrolér napísaný taktiež v jazyku Java. Je prenositeľný a umožňuje spustenie na rozličných platformách, počínajúc výkonnými multi-procesorovými systémami s operačným systémom Linux až po smartfóny s operačným systémom Android. Je uvoľnený pod licenciou GNU General Public License v2. Medzi jeho vlastnosti patrí tiež vysoký výkon, podporuje beh na viacjadrových procesoroch.

Trema
: Trema [3] je framework určený k vývoju OpenFlow kontrolérov v jazykoch C++ a Ruby. Rozširovaný je pod licenciou GNU General Public License. Je spustiteľný výlučne v prostredí GNU/Linux. Vďaka tomu, že podporuje jazyk Ruby verzie 1.8.7 uľahčuje rýchly vývoj prototypov OpenFlow kontrolérov. Po otestovaní a odladení je možné tieto kontroléry prepísať do jazyka C++. To umožní zrýchliť a zoptimalizovať

beh kontroléra. Trema v súčasnosti podporuje OpenFlow protokol verzie 1.0. Vďaka množstvu príkladov, ktoré ku frameworku Trema existujú v jazykoch Ruby aj C++, sa programátor rýchlo zorientuje a objaví možnosti, ktoré tento framework ponúka.

2.3 Trema

Pre implementáciu kontroléra v tejto bakalárskej práci som zvolil framework Trema, práve kvôli možnosti rýchleho vývoja prototypov v jazyku Ruby. Základom pri vývoji nového kontroléra postaveného na frameworku Trema v jazyku Ruby je vlastná implementácia obslužných rutín. Obslužné rutiny sú spustené pri vzniku udalosti, ktorá vyžaduje spoluprácu kontroléra. Takáto údalosť je napríklad prijatie paketu od OpenFlow switcha. Taktiež to môžu byť rôzne informatívne správy o stave zariadení alebo siete. Z obslužných rutín sú najdôležitejšie nasledovné:

packet_in (datapath_id, message)
: Obslužná rutina packet_in je spustená v okamžiku, keď OpenFlow kontrolér obdrží paket od OpenFlow switcha. To znamená, že switch nevedel klasifikovať tento paket do žiadneho dátového toku. Nenašiel teda žiadnu akciu, ktorú má na tento paket aplikovať. Parameter `datapath_id` obsahuje identifikátor OpenFlow switcha a `message` je objekt, ktorá reprezentuje samotný paket.

  OpenFlow kontrolér na základe svojho algoritmu analyzuje paket a zistí, čo je potrebné s paketom urobiť. Do Flow-Table OpenFlow switcha pridá zavolaním metódy `send_flow_mod_add()` pravidlo. Toto pravidlo obsahuje popis dátového toku, do ktorého analyzovaný paket patrí a akcie, ktoré budú pri ďalšom prechode paketu patriaceho do tohto dátového toku aplikované. Ostatné náležitosti, ktoré môže obsahovať záznam, pridávaný do Flow-Table, už boli v tomto texte spomenuté.

flow_removed (datapath_id, message)
: Táto rutina je spustená vtedy, keď na niektorom z OpenFlow switchov expiruje záznam vo Flow-Table. Tento switch je identifikovaný prostredníctvom identifikátora `datapath_id` a informácie o expirovanom toku, ako napríklad počet prenesených paketov a bajtov sú obsiahnuté v objekte `message`.

features_reply (datapath_id, message)
: Je odpoveď OpenFlow switcha na správu `features_request`. Objekt `message` obsahuje súhrnné informácie o switchi včítane informácií o jeho fyzických rozhraniach.

list_switches_reply (datapath_ids)
: Na vyžiadanie je k dispozícii zoznam všetkých zapojených OpenFlow switchov, ktoré s kontrolérom komunikujú.

openflow_error (datapath_id, message)
: Táto rutina je aktivovaná pri vzniku chybovej udalosti.

switch_ready (datapath_id)
: Oznamuje kontroléru, že sa k nemu práve pripojil nový OpenFlow switch s identifikátorom `datapath_id`.

switch_disconnected (datapath_id)
: Táto rutina oznamuje, že OpenFlow switch, ktorý bol doteraz aktívny a komunikoval s kontrolérom je odpojený. Parameter `datapath_id` identifikuje konkrétny switch.

send_flow_mod_add (datapath_id, options = {})
Pomocou tejto metódy je možné pridať nové pravidlo do Flow-table OpenFlow switcha, špecifikovaného parametrom `datapath_id`.

send_flow_mod_remove (datapath_id, options = {})
Metóda odstráni existujúce pravidlo z Flow-Table pripojeného OpenFlow switcha, špecifikovaného parametrom `datapath_id`.

send_list_switches_request
Pošle do siete požiadavku, aby sa pripojené switche identifikovali kontroléru.

send_message (datapath_id, message)
Táto metóda umožňuje kontroléru zaslať konkrétnemu OpenFlow switchu špecifickú správu, napríklad požiadavku `features_request`.

send_packet_out (datapath_id, options = {})
Metódu je možné využiť na zaslanie paketu konkrétnemu OpenFlow switchu, pričom je definovaná jednorázová akcia, čo má OpenFlow switch s týmto paketom spraviť. Takto je možné preposlať napríklad paket, ktorý je doručený kontroléru k analýze správnym smerom, aby nedošlo k jeho strate.

Ďalšie informácie o rozhraní OpenFlow frameworku Trema je možné získať v dokumentácii [2].

Kapitola 3

Jazyk OpenFlow Network Control Language (OF-NCL)

Jazyk OpenFlow Network Control Language (OF-NCL) je vysoko-úrovňový deklaratívny jazyk. Pri jeho návrhu som zakomponoval prostriedky pre popis vnútornej topológie siete a spojil ich s politikami (vysvetlené v podkapitole 3.1.9). Tie slúžia pre popis chovania siete. Kedže jazyk OF-NCL je deklaratívny, vyskytujú sa v ňom bloky kódu, na základe ktorých bude interpret riadiť počítačovú sieť. Neobsahuje priame výrazy, ktorými by mohol administrátor zasiahnuť do procesu riadenia. Pri jeho návrhu som využil aj poznatky zo Študijnej opory k predmetu IFJ [5]. V kapitole 5 predkladám čitateľovi konkrétny príklad konfigurácie sieťovej topológie pomocou navrhovaného jazyka OF-NCL.

3.1 Prvky jazyka OF-NCL

Program, zapísaný v jazyku OF-NCL sa skladá z deklarácií entít. Každú entitu je možné rozšíriť pomocou bloku špecifikácií. Blok špecifikácií obsahuje atribúty, ktoré opisujú konkrétnu entitu alebo kladú požiadavky na správanie tejto entity. Každá entitá má vlastné atribúty. Niektoré z nich sú povinné a nevyhnutné k správnemu procesu riadenia počítačovej siete, iné sú voliteľné. Atribúty sa môžu vzájomne dopĺňať a ovplyvňovať. V zdrojovom kóde je možné využívať riadkové komentáre. Jazyk zahŕňa aj možnosť zjednodušiť a sprehľadniť zápis programu pomocou makier. Tieto sú opísané v nasledujúcom texte.

3.1.1 Základné črty jazyka

Jazyk OpenFlow Network Control Language rozlišuje veľkosť znakov. Nevyžaduje striktné odsadzovanie a biele znaky slúžia k oddeleniu príkazov a operátorov v konštrukciách tohto jazyka. Napriek tomu sa odporúča používať vhodné odsadzovanie, ktoré výrazne sprehľadní zdrojové kódy. Pri deklaráciách sa vytvárajú entity, ktoré sú jednoznačne pomenované. Ich symbolický názov vzniká pri deklarácii a musí byť jedinečný (deklarovať dve entity s rovnakým názvom je neprípustné aj v prípade, že sa jedná o entity rôzneho typu). Programátor môže používať prefixy pomocou podtržítka pre odlíšenie typu entít.

3.1.2 Riadkové komentáre

Jazyk podporuje riadkové komentáre. Za komentár sa považuje každý text, zapísaný za dvojicou znakov '//', podobne ako v jazyku C. Komentár končí koncom riadku. Komentáre

sú vhodné na doplnenie zápisu v programovacom jazyku o vysvetlenie, ktoré môže zjednodušiť neskoršie úpravy alebo rozšírenia v programe. Riadkový komentár môže vyzerať takto:

```
// toto je komentár
```

3.1.3 Makro #include

Toto makro slúži k zahrnutiu externých súborov do hlavného zdrojového kódu. Zdrojový kód z externého súboru sa doplní na miesto, kde sa vyskytuje macro #include. Takto je umožnené členenie zdrojového kódu a jeho rozdelenie na menšie časti. Príklady použitia:

```
#include /home/user/sources/defines.ofncl
#include ../defines.ofncl
```

3.1.4 Makro #define

Makro #define umožňuje zadefinovať symbolický názov k reťazcu, ktorý sa v zdrojovom kóde často vyskutuje. Pomocou symbolického názvu je potom možné odkazovať sa na definovaný reťazec. Symbolický názov musí byť zapísaný veľkými písmenami, akceptovaný je aj znak '_' (podtržítko). Symbolický názov je pri každom ďalšom použití nahradený k nemu prislúchajúcim reťazcom. Zadefinované reťazce sa ukladajú do tabuľky a je možné odkazovať sa pomocou nich aj v externých súboroch, vložených do zdrojového kódu pomocou makra #include.

3.1.5 Deklarácia SWITCH

V zdrojovom kóde sa takto deklaruje fyzický OpenFlow switch, ktorý bude pracovať na základe pravidiel, nastavených OpenFlow kontrolérom. Povinný atribút v tomto bloku je **DATAPATH**. Hodnota nastavená pomocou tohto atribútu slúži k jednoznačnej identifikácii fyzického zariadenia v sieťovej topológii. Operátor priradenia '=' je voliteľný. Gramatické pravidlá pre entitu switch a tento atribút:

```
<switch> → SWITCH <switch_name> <specs>;
<specs> → EPSILON
<specs> → { <specs_line> }
<specs_line> → <attribute> \n <specs_line>
<specs_line> → EPSILON

<attribute> → DATAPATH <datapath_id>
```

Ďalším atribútom je **INTERFACE**, ktorý vytvorí rozhranie pre pripojenie koncových zariadení, napríklad pracovných staníc alebo serverov. K tomuto rozhraniu sú priradené fyzické porty OpenFlow switcha. Umožňuje priradiť k rozhraniu jeden alebo viac fyzických portov. Gramatické pravidlá pre tento atribút:

```
<attribute> → INTERFACE <interface_name> : <port_range>
<port_range> → <start> <upto>
<upto> → EPSILON
<upto> → - <end_port>
```

Podobne ako predchádzajúci funguje aj atribút **INTERCONN**. Tento sa používa k vytvoreniu rozhrania pre komunikáciu s iným OpenFlow switchom. Vyžaduje priradenie práve jedného fyzického rozhrania na OpenFlow switchi. Gramatika je:

```
<attribute> → INTERCONN <interface_name> : <port_number>
```

Dôležitým atribútom je **CONTROL**. Tento sa používa pre priradenie Policies (budú vysvetlené v ďalšom texte) do konfigurácie switcha. OpenFlow switch pracuje iba na základe priradených Policies. Je vhodné zdôrazniť, že ak OpenFlow switch nedostane v deklarácii priradenú žiadnu Policy, implicitne zahodí všetky dátové toky. Gramatika atribútu:

```
<attribute> → CONTROL <policy_name> <policies>
<policies>  → , <policy_name> <policies>
<policies>  → EPSILON
```

3.1.6 Deklarácia LINK

Vyjadruje existenciu fyzického spojenia medzi dvoma OpenFlow switchmi. Takéto spojenie vytvára linku medzi zariadeniami. Poznať linky v topológii OpenFlow siete je nevyhnutné kvôli rýchlemu a efektívnemu vytvoreniu spojenia medzi dvoma koncovými stanicami. Samotná deklarácia linky nevyžaduje použitie bloku atribútov. Linku je však možné pomocou atribútu **COST** vhodne oceniť. Táto cena sa môže neskôr uplatniť pri výpočte a určení trasy spojenia pre dátový tok pomocou grafového algoritmu. Nasleduje gramatika pre deklaráciu linky:

```
<link> → LINK <link_name> BETWEEN <device_name>.<device_interconn> AND
<device_name>.<device_interconn> <specs>;
<specs> → EPSILON
<specs> → { <specs_line> }
<specs_line> → <attribute> \n <specs_line>
<specs_line> → EPSILON
<attribute> → COST = <cost_value>
```

3.1.7 Deklarácia PATH

Umožňuje špecifikovať parametre trasy dátového toku pri jej výpočte. Použitím tejto deklarácie sa vytvorí nová entita Path, ktorá môže obsahovať atribút **WAYPOINT**. Tento atribút umožňuje nastaviť zariadenia (OpenFlow switche), ktorými bude dátový tok prechádzať. Táto možnosť je zaujímavá napríklad v prípade, že chceme odľahčiť vyťažené zariadenia, alebo naopak presunúť záťaž na zariadenia, ktoré majú dostatočnú kapacitu. Zaujímavé môže byť tiež smerovanie sieťových tokov cez zariadenie, ku ktorému je pripojená

sonda na zber údajov o prenesených dátach. Dôležité dátové toky je možné týmto spôsobom ochrániť aj pred prechodom menej bezpečnými miestami siete, napríklad segmentami, ktoré sú verejne prístupné alebo ktoré môžu byť cieleným útokom typu DoS vyradené z prevádzky. Gramatika Path:

```
<path> → PATH <path_name> <specs>;
<specs> → EPSILON
<specs> → { <specs_line> }
<specs_line> → <attribute> \n <specs_line>
<specs_line> → EPSILON
<attribute> → WAYPOINT <wp_device> <wps>
<wps> → EPSILON
<wps> → , <wp_device> <wps>
```

3.1.8 Deklarácia FLOW

Dátový tok je postupnosť paketov so spoločnou vlastnosťou. Práve v deklarácii dátového toku sa dá popísať zamýšľaný tok pomocou parametrov, ktoré sú pre pakety tohto dátového toku špecifické.

```
<flow> → FLOW <flow_name> <specs>;
<specs> → EPSILON
<specs> → { <specs_line> }
<specs_line> → <attribute> \n <specs_line>
<specs_line> → EPSILON
```

SRC.IP

Zdrojová IP adresa. Paket patrí do tohto dátového toku, ak sa s touto IP adresou zhoduje jeho zdrojová IP adresa. Umožňuje zadať jednu IP adresu alebo rozsah IP adries. Pri zadaní rozsahu IP adries musí zdrojová IP adresa paketu byť väčšia alebo rovná prvej z dvojice zadaných adries a menšia alebo rovná druhej z dvojice zadaných adries. Gramatika tohto atribútu je:

```
<attribute> → SRC.IP = <ip_start> <ip_range>
<ip_range> → EPSILON
<ip_range> → - <ip_end>
```

SRC.PN

Číslo portu na zdrojovej stanici. Je možné zadať aj rozsah portov. Číslo portu alebo rozsah musí byť v intervale 0 - 65535. Gramatika:

```
<attribute> → SRC.PN = <pn_start> <pn_range>
<pn_range> → EPSILON
<pn_range> → - <pn_end>
```

DST.IP

Cieľová IP adresa. Funguje analogicky ako zdrojová IP adresa a umožňuje zadať rozsah IP adries. Oproti zdrojovej IP adrese sa mení prvé pravidlo gramatiky na:

```
<attribute> → DST.IP = <ip_start> <ip_range>
```

DST.PN
Číslo portu na cieľovej stanici. Umožňuje zadať rozsah portov. Číslo portu alebo rozsah musí byť v intervale 0 - 65535. Gramatiku rozširuje o pravidlo:

```
<attribute> → DST.PN = <pn_start> <pn_range>
```

PT
Protokol použitý na vrstve L3 alebo L4. Podporuje protokoly ICMP, TCP a UDP. Pri protokole ICMP neumožňuje kombinovať s atribútmi SRC.PN a DST.PN. Gramatické pravidlá:

```
<attribute> → PT = <protocol> <pts>
<pts> → , <protocol> <pts>
<pts> → EPSILON
```

3.1.9 Deklarácia POLICY

Funkcionalita OpenFlow sietí je zabezpečená pomocou politík. Každá politika je postupnosť pravidiel s rôznymi vlastnosťami. Na základe niektorého z pravidiel môže byť potom vytvorené spojenie medzi dvoma koncovými stanicami. Poznáme dva typy pravidiel:

FLOW
Tento typ umožňuje vytvorenie spojenia na základe popisu dátového toku. Keďže dátový tok považujeme za jednosmerný, komunikácia podľa tohto pravidla bude akceptovaná iba jedným smerom.

CONVERSATION
Na rozdiel od predchádzajúceho, tento typ pravidiel akceptuje okrem dátového toku, ktorý inicioval komunikáciu, aj tok, ktorý nesie odpoveď. Vytvorí sa teda konverzácia medzi dvoma koncovými stanicami.

Každé pravidlo v OpenFlow politike musí špecifikovať trasu v sieťovej topológii (entita Path). Podľa nej sa vytvorí spojenie. Pokiaľ nie je vyžadované striktné smerovanie cez konkrétne segmenty siete, je možné namiesto trasy použiť zástupný znak '*'. Teraz bude uvedená gramatika Policy:

```
<policy> → POLICY <policy_name> <specs>;
<specs> → EPSILON
<specs> → { <specs_line> }
<specs_line> → EPSILON
<specs_line> → <attribute> \n <specs_line>

<attribute> → <rule_type> <flow_name> PATH <path_name> <rule_specs>
<rule_type> → FLOW
<rule_type> → CONVERSATION

<rule_specs> → EPSILON
<rule_specs> → { <rule_specs_line> };
<rule_specs_line> → EPSILON
```

```
<rule_specs_line> → <attribute> \n <rule_specs_line>

<attribute> → GRAPH <graph_name>
<attribute> → TIMEOUT-HARD <time>
<attribute> → TIMEOUT-IDLE <time>
```

Každé pravidlo v Policy je možné rozšíriť blokom špecifikácií o atribúty. Atribút **GRAPH** špecifikuje funkciu, ktorá bude použitá pri výbere najvhodnejšej cesty. Jazyk umožňuje vytvorenie vlastných funkcií, ako uvádzam v nasledujúcej kapitole.
Atribút **TIMEOUT-HARD** umožňuje nastaviť, ako dlho bude dátový tok po vytvorení existovať. Atribút **TIMEOUT-IDLE** umožňuje nastaviť, ako dlho bude dátový tok existovať po tom, čo sieťou prejde posledný paket, zaradený do tohto toku. Časový údaj je pritom možné zapísať prostredníctvom celého čísla a jednopísmenovej skratky, ktorá označuje časovú jednotku:

s počet sekúnd

m počet minút

h počet hodín

d počet dní

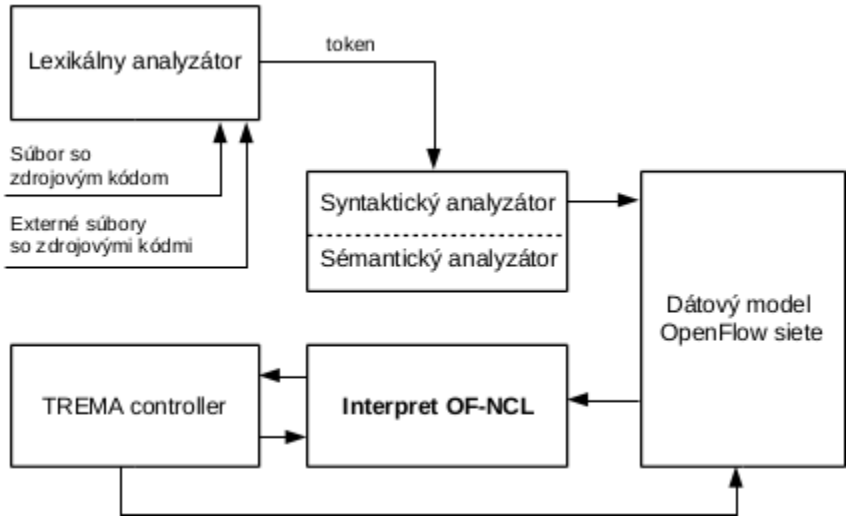
Časový údaj desať sekúnd bude zapísaný '`10s`', dvadsaťpäť minut sa zapíše '`25m`' a dva dni '`2d`'.

Kapitola 4

Implementácia interpretu jazyka OF-NCL

Využitie frameworku, ktoré poskytuje Trema OpenFlow kontrolér v jazyku Ruby umožňuje skutočne rýchly vývoj prototypov vlastných OpenFlow kontrolérov. Kvôli zachovaniu kompatibility s kontrolérom Trema bol celý interpret implementovaný v jazyku Ruby verzia 1.8.7. Aplikácia je spustiteľná v prostredí operačného systému Linux. Odporúčaná distribúcia je Linux Ubuntu vo verzii 12.04 alebo vyššej.

Na začiatku tejto kapitoly predkladám spôsob implementácie a konceptuálny model prekladača jazyka OF-NCL (obrázok 4.1). Z programu, zapísaného v tomto jazyku sa vytvára dátový model OpenFlow siete. Tento je predstavený v podkapitole 4.3. Na základe vygenerovaného dátového modelu je interpret schopný riadiť OpenFlow sieť. Samotný proces riadenia siete je vysvetlený v podkapitole 4.7. Dôležitou vlastnosťou v tejto implementácii je navrhnutá metóda značkovania aktívnych dátových tokov. Vysvetlená je v podkapitole 4.5. Jej súčasťou je aj tabuľka Active-Flow (podkapitola 4.4). Pri samotnom behu interpretu si tento vytvára interné mapovanie koncových zariadení. Využíva k tomu protokol ARP. Tento proces má na starosti programová komponenta ARP-Manager, popísaná v kapitole 4.6. Toto mapovanie je tiež súčasťou dátového modelu siete.



Obrázek 4.1: Schematické zobrazenie komponent interpretu jazyka OF-NCL

Vstupom pre interpret jazyka OpenFlow Network Control Language je súbor so zdrojovým kódom v tomto jazyku. Tento súbor sa načíta a interpret spracuje programové konštrukcie v ňom zapísané.

4.1 Lexikálna analýza zdrojového kódu

Prvou fázou prekladu zdrojového kódu, ktorá nasleduje bezprostredne po načítaní súboru so zdrojovým kódom je lexikálna analýza. Lexikálna analýza spočíva v rozpoznaní lexémov v zdrojovom kóde. Pri každom lexéme sa určí jeho typ a zaradí sa do fronty, z ktorej bude neskôr dostupný pre syntaktický analyzátor.

4.1.1 Postup lexikálnej analýzy

Na začiatku je súbor so zdrojovým kódom rozdelený na podreťazce. Rozdelenie prebieha na základe pravidiel, špecifických pre jazyk OF-NCL. Rozpoznávajú sa nasledujúce druhy lexémov:

Token KEY
: Označuje kľúčové slová jazyka OF-NCL ako **SWITCH**, **INTERCONN** a iné. Kľúčové slová jazyka nie je možné používať ako symbolické názvy ani v definíciách. Kľúčové slová jazyka OF-NCL sú uvedené v prílohe B.

Token OPERATOR
: Tento typ tokenov zahŕňa všetky operátory, ktoré jazyk obsahuje. Sú to okrúhle zátvorky '(' a ')', zložená zátvorky '{' a '}', bodkočiarka ';', dvojbodka ':', lomítko '/', bodka '.', čiarka ',', znamienko rovnosti '=' a pomlčka '-'.

Token VALUE
: Všetky ostatné tokeny sú zaradené do tejto kategórie. Sú to symbolické názvy a parametre deklarovaných entít.

4.1.2 Spracovanie makier

Ešte predtým, než začne lexikálny analyzátor rozpoznávať jednotlivé lexémy, je potrebné spracovať makrá `#include` a `#define`.

Makro **#include** obsahuje jeden parameter - relatívnu alebo absolútnu cestu k externému súboru so zdrojovým kódom. Makro #include musí byť spracované ako prvé, pretože zahŕňa externé súbory, ktoré môžu obsahovať ďalšie makrá. Obsah externého súboru so zdrojovým kódom, načítaný pomocou makra #include, je doplnený presne na miesto, kde sa vyskytovalo toto makro. Makro teda expandovalo na obsah externého súboru, ktorý odkazovalo.

Nasleduje spracovanie makier **#define**. Toto makro vyžaduje dva parametre. Prvým je symbolický názov definície. Pomocou tohto názvu je možné kedykoľvek v ďalšom zdrojovom kóde odkazovať reťazec, ktorý je zadaný ako druhý parameter. Takto zadefinovaný reťazec je možné odkazovať aj z externých súborov, zahrnutých pomocou makra #include.

4.1.3 Trieda Tokenizer - rozdelenie zdrojového kódu na lexémy

Zdrojový kód je teraz spracovaný po riadkoch. Odstránia sa riadky, na ktorých sú zapísané makrá. Makro začína na riadku vždy prvým znakom '#'. Regulárny výraz '`^#.*`' pokryje všetky makrá v zdrojovom kóde.

Zo zdrojového kódu sa odstránia aj všetky riadkové komentáre. Riadkové komentáre sú pokryté pomocou regulárneho výrazu '`\/\/.*$`'.

Teraz sa zdrojový kód transformuje na postupnosť tokenov. Základným oddelovačom lexémov v syntaktických konštrukciách jazyka sú biele znaky. Skupiny bielych znakov sa nahradia znakom medzera ' '. To docielime volaním metódy `gsub(/\t\r\f+/, ' ')` nad pôvodnými úsekmi zdrojového kódu.

Keďže operátory nemusia byť nutne oddelené od okolitých konštrukcií bielym znakom, pridá sa pred a za každý operátor chýbajúca medzera. Takto upravený zdrojový kód sa rozdelí na pole tokenov volaním metódy `split(' ')`.

Vyššie opísanú funkcionalitu implementuje trieda Tokenizer v súbore *tokenizer.rb*.

4.1.4 Trieda TokenTable - tabuľka tokenov

Táto trieda je implementovaná v súbore *token.rb*. Zapúzdruje modifikovanú frontu a metódy, pomocou ktorých je možné posúvať sa vo fronte ľubovolným smerom. Táto fronta obsahuje tokeny. Každý token je inštanciou podľa triedy Token, ktorá je tiež implementovaná v tomto súbore. Každá inštancia triedy Token je charakterizovaná svojim typom a reťazcom tokenu. Trieda TokenTable implementuje metódy:

prevToken()
: Posunie ukazateľ v tabuľke tokenov na predchádzajúci token. Ak tento ukazuje na prvý token vo fronte, pozícia sa nezmení.

nextToken()
: Posunie ukazateľ v tabuľke tokenov na nasledujúci token. Ak tento ukazuje na posledný token vo fronte, pozícia sa nezmení.

getToken()
: Vráti hodnotu aktuálneho tokenu, ktorý je odkazovaný ukazateľom.

insert(token)
: Zaradí nový token na koniec fronty.

is_end()
: Prediktor, ktorý nadobudne hodnotu *true*, ak ukazateľ ukazuje na posledný token v tabuľke.

4.2 Syntaktická a sémantická analýza

Syntaktický analyzátor implementuje trieda **Parser** v súbore *parser.rb*. Syntaktická analýza sa spúšťa zavolaním metódy `run()`. Realizovaná je metódou rekurzívneho zostupu. Na základe gramatiky (príloha A) vykonáva postupne syntaktickú analýzu zhora nadol. Kedže sa jedná o syntaxou riadený preklad, syntaktický analyzátor vždy vo vhodnej chvíli volá časť lexikálneho analyzátoru nazývanú tokenizer, popísaný v kapitole 4.1.4. Tokenizer udržiava tabuľku tokenov, načítaných zo zdrojového kódu. Volaním metódy `getToken()`

získa syntaktický analyzátor nový token, ktorý je zároveň nastavený ako aktívny. Posun na nasledujúci token je možný volaním metódy `nextToken()`.

Syntaktický analyzátor na základe typu a hodnoty získaného tokenu rozhodne o ďalšej akcii. Možno povedať, že v každom stave syntaktického analyzátoru existuje množina očakávaných tokenov. Ak táto množina obsahuje aktuálny token, zavolá sa metóda, ktorá reprezentuje obsluhu pre tento token. Zavolaním tejto metódy sa syntaktický analyzátor dostane do nového stavu. Ak sa aktuálny token v množine očakávaných tokenov nenachádza, jedná sa o syntaktickú chybu. Program je ukončený a užívateľ obdrží na štandardnom chybovom výstupe hlásenie o syntaktickej chybe.

Sémantická analýza bola implementovaná priamo do syntaktického analyzátoru. Hlavnou úlohou sémantickej analýzy je kontrola vstupných hodnôt, ktoré nemožno pri syntaktickej analýze overiť. Kontrolu je možné vykonať až na základe načítaných hodnôt. Pre ilustráciu, sémantická chyba môže byť duplicitné pomenovanie novej deklarácie entity, pričom už existuje iná entita s takýmto názvom. Za sémantickú chybu sa považuje aj nekorektný zápis IP adresy (napríklad 100.200.300.400) alebo číslo portu TCP/UDP protokolu mimo povolený rozsah.

Súčasťou syntaktickej analýzy je vytváranie dátového modelu siete na základe deklarácií, zapísaných v zdrojovom kóde. Každý typ entity je v dátovom modeli reprezentovaný vlastnou triedou. V diskutovanom interprete existujú nasledujúce triedy, ktoré reprezentujú entity:

VSwitch
: OpenFlow Switch (entita popísana v kapitole 3.1.5)

VLink
: linka medzi dvoma OpenFlow switchmi (entita popísana v kapitole 3.1.6)

Path
: požiadavky na trasu pri naväzovaní spojenia (entita popísana v kapitole 3.1.7)

VFlow
: dátový tok, popis parametrov hlavičiek paketu (entita popísaná v kapitole 3.1.8)

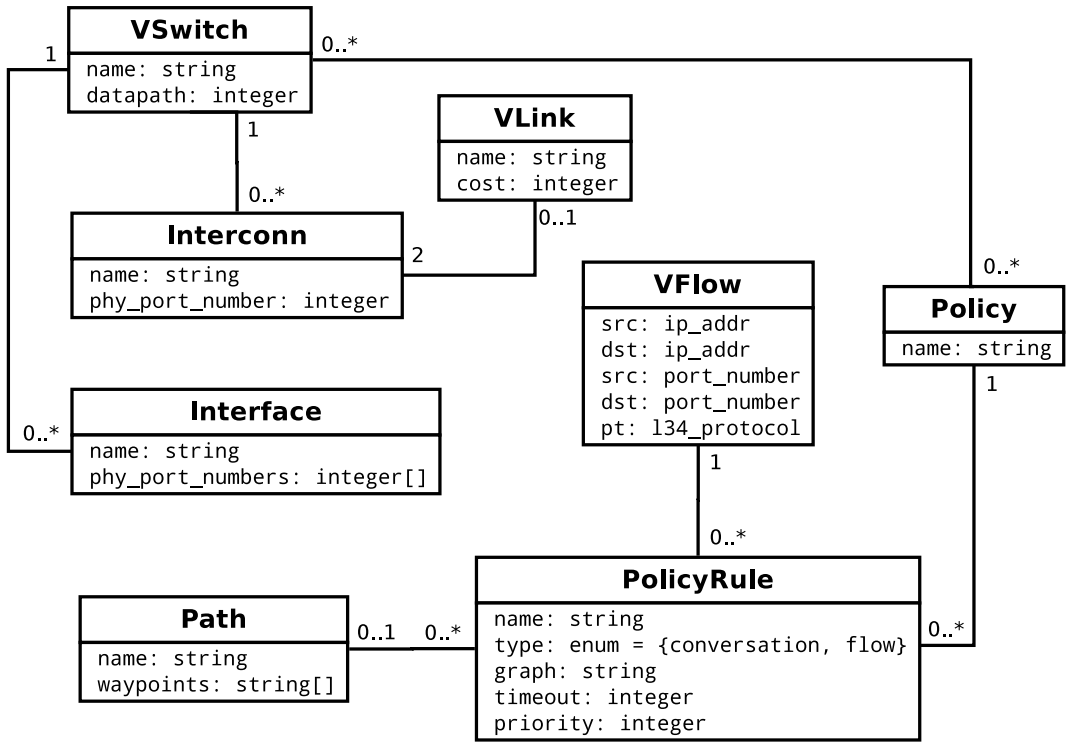
Policy
: politika (entita popísaná v kapitole 3.1.9)

4.3 Dátový model OpenFlow siete

Dátový model je realizovaný formou verejného, globálne prístupného hashovaného pola `$devices`. V ňom sú uložené inštancie tried, ktoré reprezentujú príslušné entity, ako je znázornené v diagrame na obrázku 4.2. Pristúpiť ku konkrétnej entite je možné cez index, ktorý zodpovedá jej názvu. Okrem toho existujú v dátovom modeli aj ďalšie polia, kde sú uložené referencie na niektoré často pristupované inštancie tried. Tieto sú indexované podľa niektorého ich špecifického parametru, čo výrazne urýchľuje prístup k nim. Napríklad OpenFlow switche sú úložené v hashovanom poli s názvom `$switches`. V rámci neho sú indexované na základe Datapath identifikátora[1].
Po skončené syntaktickej analýzy je k dispozícii základný dátový model siete podľa konfigurácie, zadanej v zdrojovom súbore. Na základe tohto dátového modelu je možné riadiť

[1] Datapath identifikátor jednoznačne identifikuje switch v rámci OpenFlow topológie

komunikáciu na sieti. Počas behu programu dochádza k interakcii s Trema kontrolérom popísaným v časti 2.3. Tento modifikuje objekty dátového modelu na základe aktuálneho stavu siete, čo je veľmi žiadúce a významné v OpenFlow sieťach. Na základe údajov, ktoré sú prístupné cez dátový model, je možné vypočítať a určiť tok dát OpenFlow sieťou. Zmeny týchto údajov sú sledované. Ak dôjde k zmene, ktorá ovplyvňuje práve prebiehajúci reláciu, môže byť bezodkladne vykonaná úprava. Výsledkom môže byť zachovanie existujúcej relácie a minimalizácia výpadkov spôsobených poruchami. Dátový model je možné jednoducho rozšíriť a pridať tak interpretu nové vlastnosti.



Obrázek 4.2: E-R diagram znázorňujúci dátový model OpenFlow siete

4.4 Active-Flow tabuľka

Active-Flow tabuľka je súčasťou dátového modelu OpenFlow siete. Každý záznam v tejto tabuľke zodpovedá jednému dátovému toku, ktorý prechádza sieťovou topológiou. Záznam v tabuľke sa skladá z položiek:

- Špecifická MAC adresa toku
- MAC adresa vysielajúcej stanice
- MAC adresa prijímajúcej stanice
- Cesta v topológii - postupnosť OpenFlow switchov, ktorými dátový tok prechádza
- Popis parametrov paketu, špecifického pre tento dátový tok

- Časový údaj, ktorý hovorí ako dlho bude tento aktívny dátový tok existovať
- Iné doplňujúce údaje kontroléra

Záznam v tabuľke Active-Flow vytvára kontrolér na základe spracovania prijatého paketu, ktorý OpenFlow switch nebol schopný sám spracovať. Paket, ktorý prechádza OpenFlow topológiou, je možné jednoducho zaradiť a nájsť zodpovedajúci záznam v tabuľke Active-Flow na základe špecifickej Flow-MAC adresy. Ako bude vysvetlené v nasledujúcej kapitole, táto MAC adresa je jedinečná a slúži pre zaradenie paketu do známeho toku.

4.5 Metóda označovania dátových tokov

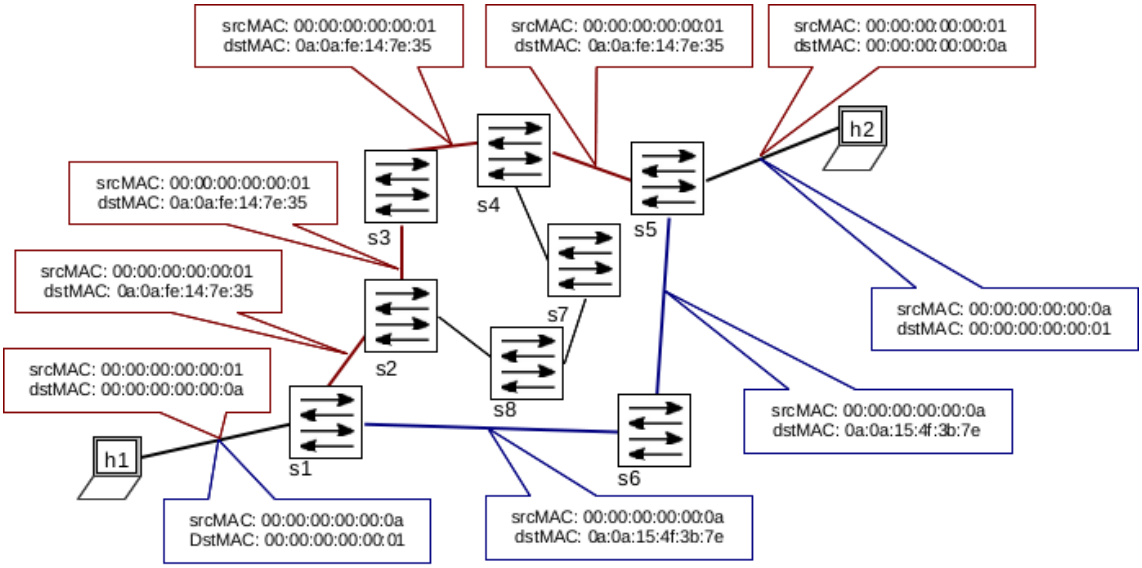
Dátový tok je postupnosť paketov so spoločnou vlastnosťou. Pri prechode dátového toku sieťou je žiaduce, aby bolo možné vždy jednoznačne identifikovať, ku ktorému dátovému toku patrí konkrétny paket. V tejto práci bol navrhnutý systém značkovania. To je realizované umiestnením špecifickej MAC adresy do poľa cieľovej MAC adresy paketu. Táto MAC adresa je náhodne vygenerovaná a využíva voľný rozsah MAC adries, aby nedošlo ku kolízii s inou, existujúcou MAC adresou reálneho hardwarového zariadenia.

Záznam o každom dátovom toku je umiestnený v tabuľke Active-Flow (tabuľka 4.1). Súčasťou tejto tabuľky je aj zdrojová a cieľová MAC adresa koncových zariadení, ktoré medzi sebou komunikujú. Cieľová MAC adresa sa nahradí MAC adresou, ktorá je špecifická pre konkrétny dátový tok. Táto zmena sa vykonáva na prvom OpenFlow switchi na ceste, ku ktorému je pripojené vysielajúce zariadenie. Do OpenFlow topológie je teda vyslaný paket so špecifickou MAC adresou, na základe ktorej je možné okamžite zaradiť paket do existujúceho dátového toku. Keď paket príde na posledný OpenFlow switch (za ktorým nasleduje prijímajúce koncové zariadenie), zmení sa cieľová MAC adresa späť na MAC adresu prjímajúceho koncového zariadenia. Tento spôsob je znázornený na obrázku 4.3.

Pre ilustráciu bude popísaný tok, označený v tabuľke 4.1 číslom 2 a v obrázku 4.3 modrou farbou. Pakety z tohto toku sú vysielané stanicou `h2`. Po prijatí paketu na switch `s5` zmení tento na základe internej Flow-Table cieľovú MAC adresu paketu na adresu, ktorá je špecifická pre tento dátový tok. V tomto prípade sa zmení adresa `00:00:00:00:00:01` na adresu `0a:0a:15:4f:3b:7e`. Takto upravený paket sa odošle na switch `s6`. Switch `s6` paket nemodifikuje a prepošle ho na switch `s1`. K tomuto switchu je pripojená stanica, pre ktorú je paket určený. Switch `s1` modifikuje cieľovú MAC adresu na skutočnú MAC adresu stanice `h1` a prepošle jej tento paket.

#	Flow-MAC	Source MAC	Destination MAC	Path
1	0a:0a:fe:14:7e:35	00:00:00:00:00:01	00:00:00:00:00:0a	s1, s2, s3, s4, s5
2	0a:0a:15:4f:3b:7e	00:00:00:00:00:0a	00:00:00:00:00:01	s5, s6, s1

Tabulka 4.1: Active-Flow - tabuľka aktívnych dátových tokov



Obrázek 4.3: Znázornený systém značkovania dátových tokov

4.6 ARP-Manager

Táto programová komponenta slúži k podpore ARP protokolu prevádzkovaného v sieťach. ARP[2] slúži k prekladu IP adries na MAC adresy. Tieto sú potrebné pre nižšie vrstvy OSI modelu k doručeniu paketov na cieľovú stanicu. Implementovaný ARP-Manager sleduje správy ARP v OpenFlow sieti a zefektívňuje fungovanie tohto protokolu. V dátovom modeli udržiava vlastnú internú ARP-cache[3] pamäť. Keď zaregistruje novú správu typu ARP-Request, uloží si dotazovanú IP adresu a rozšíri túto správu na všetky pripojené koncové zariadenia v OpenFlow topológii. Zároveň si do internej ARP-cache pamäti uloží aj preklad IP adresy na MAC adresu pre stanicu, ktorá ARP-Request zaslala.

Stanica, odpovedajúca správou ARP-Reply vyšle túto správu, ktorú zaregistruje kontrolér. Ak kontrolér zistí, že má vo svojej internej ARP-cache pamäti nekompletný záznam, doplní k príslušnej IP adrese MAC adresu podľa ARP-Reply. Zároveň správu ARP-Reply prepošle stanici, ktorá o preklad požiadala.

Kontrolér naviac do svojej internej ARP-cache pamäti uloží aj informáciu o tom, ku ktorému OpenFlow switchu je koncová stanica pripojená a takisto číslo fyzického portu. Táto informácia je neskôr využívaná pri výpočte trasy pred ustanovením dátového toku.

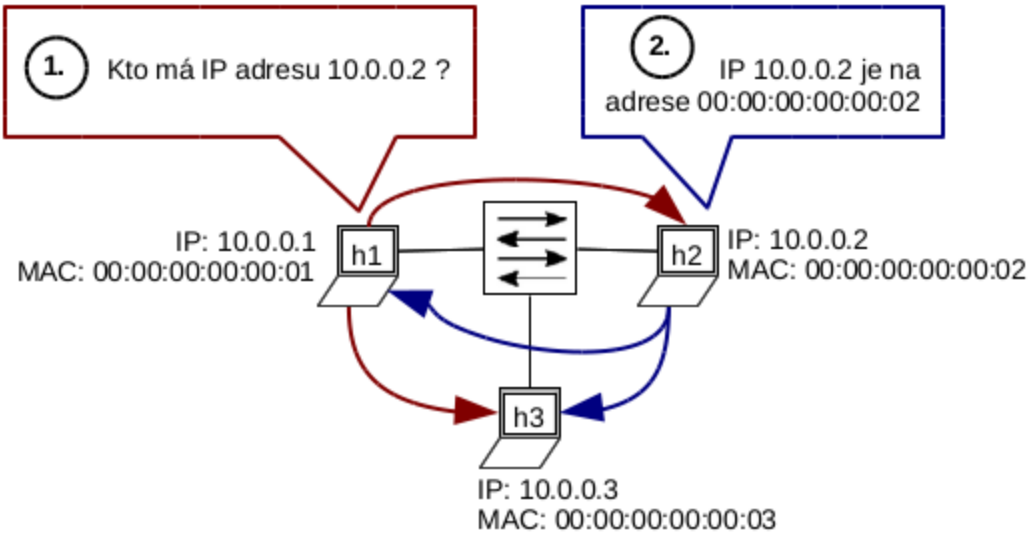
Veľmi výhodné je, že v momente, keď iná stanica požaduje preklad IP adresy na MAC adresu a tento preklad sa už nachádza v internej ARP-cache pamäti kontroléra, ten môže okamžite vygenerovať a zaslať ARP-Reply stanici, ktorá preklad požadovala. Týmto sa celý ARP preklad urýchli a sieťou sa nešíria prebytočné broadcastové správy.

Dá sa predpokladať, že táto technika dokáže do značnej miery urýchliť a zefektívniť ARP preklad.

V tejto súvislosti treba však spomenúť aj možnú zmenu MAC adresy stanice, ktorá sa už vyskytuje v internej ARP-cache pamäti kontroléra. Taktiež môže dôjsť k tomu, že na stanici

[2] Address Resolution Protocol
[3] Tabuľka prekladov IP adries na MAC adresy.

bude zmenená IP adresa a v ARP-cache sa budú vyskytovať dva záznamy s rozličnými IP adresami a zhodou v MAC adrese. Čiastočne je možné tento stav eliminovať obmedzením doby životnosti ARP záznamov v internej ARP-cache. Taktiež je možné robiť periodickú kontrolu záznamov, ktoré sa v tejto pamäti nachádzajú. Takéto situácie a prípadné konflikty by mohla riešiť vyššia softvérová vrstva, ktorá bude s interpretom OF-NCL a zahrnutým dátovým modelom siete komunikovať prostredníctvom poskytovaného API rozhrania.



Obrázek 4.4: ARP dotazovanie

ARP dotaz posiela stanica, ktorá chce vysielať dáta, ale nepozná MAC adresu cieľovej stanice. ARP dotaz "Kto má IP adresu XXX.XXX.XXX.XXX ?" vyšle broadcastom do siete a čaká na odpoveď.

Stanica, ktorá zaregistruje ARP požiadavku na preklad jej vlastnej IP adresy odpovedá správou: "IP XXX.XXX.XXX.XXX je na adrese XX:XX:XX:XX:XX:XX.". Túto správu odošle na broadcast.

4.7 Proces riadenia OpenFlow siete

Princíp OpenFlow sietí bol vysvetlený v kapitole 2.1. Samotný proces riadenia pozostáva z dotazovania sa kontroléra na pakety, ktoré OpenFlow switch nie je schopný spracovať samostatne na základe pravidiel v jeho Flow-table (kapitola 2.1.2). Prijatie paketu kontrolérom je obslúžené handlerom `packet_in()`. Ak bol prijatý paket, ktorý obsahuje dáta protokolu ARP, je tento paket odovzdaný ARP-manageru, ktorého princíp je vysvetlený v kapitole 4.6. Ak sa jedná o iný paket, pochádzajúci z bežnej komunikácie, je zahájená obsluha volaním metódy `packetRoutine(datapath_id, message)`. Prvý parameter `datapath_id` označuje identifikátor OpenFlow switcha. Druhý parameter `message` nesie samotný paket. Tento paket je reprezentovaný triedou `PacketIn`. Z paketu sú extrahované hlavičkové informácie z rôznych vrstiev OSI modelu. Kontrolér na základe cieľovej MAC adresy tohto paketu zistí, či táto MAC adresa nereprezentuje špecifickú Flow-MAC adresu. Ak áno, je zrejmé, že sa jedná o existujúci dátový tok na ceste OpenFlow topológiou. Záznam o ňom sa teda vyskytuje v tabuľke Active-Flow (popísaná v kapitole 4.4).

4.7.1 Nastavenie existujúceho dátového toku v OpenFlow zariadení

Z tabuľky Active-Flow, uloženej v dátovom modeli kontroléra, sú vytiahnuté informácie o aktuálne spracovávanom dátovom toku. Najdôležitejšia informácia je cesta, ktorá je pre tento dátový tok vyhradená. Najskôr kontrolér overí, či sa OpenFlow switch, na ktorý prišiel tento paket, vyskytuje v ceste vyhradenej pre dátový tok. Ak sa tam nenachádza, pravdepodobne sa jedná o chybu pri deklarácii liniek medzi OpenFlow switchmi, prípadne o konflikt s týmito deklaráciami a skutočným fyzickým zapojením sieťovej topológie. Ak je všetko v poriadku, OpenFlow switch sa v tejto ceste vyskytovať bude. Kontrolér v tejto chvíli zaujíma, aké je nasledujúce zariadenie za týmto OpenFlow switchom. Existujú dve možnosti. Prvá možnosť je, že cesta definuje ďalší OpenFlow switch. Vtedy kontrolér zistí, ktorou linkou je aktuálny OpenFlow switch pripojený k nasledujúcemu. Potom vygeneruje nové pravidlo, ktoré bude nastavené do Flow-Table tohto OpenFlow switcha. Toto pravidlo bude špecifikovať dátový tok a číslo fyzického portu switcha, na ktorý sa pakety tohto dátového toku budú preposielať. Druhá možnosť je, že toto zariadenie je posledným OpenFlow zariadením v ceste tohto dátového toku. To znamaná, že k tomuto OpenFlow switchu je pripojená priamo koncová stanica, pre ktorú je tento dátový tok určený. Ak táto koncová stanica bola medzičasom odpojená, prípadne presunutá, switch dáta zahodí. Pravdepodobne však koncová stanica bude k tomu switchu pripojená na rozhranie typu `INTERFACE`, ktoré je definované aj v zdrojovom programe. Kontrolér vygeneruje pravidlo, na základe ktorého OpenFlow switch prepošle dáta na fyzické rozhranie OpenFlow switcha, ku ktorému je pripojená koncová stanica. Toto pravidlo obsahuje ešte jednu akciu, ktorá zabezpečí zmenu cieľovej MAC adresy. Táto sa prepíše z tzv. Flow-MAC adresy, špecifickej pre dátový tok na skutočnú MAC adresu cieľovej stanice.

4.7.2 Ustanovanie nového dátového toku do tabuľky Active-Flow

Ďalšia možnosť je, že informácia o tomto dátovom toku sa v tabuľke Active-Flow nenachádza. Znamená to, že paket ešte nebol preposlaný žiadnym OpenFlow switchom a bol prijatý od vysielajúcej stanice. Nasledujú operácie, ktoré zistia, ako sa má naložiť s týmto paketom. Paket môže byť odpoveďou na iný dotazovací tok. Najskôr sa kontrolér pokúsi overiť, či takýto dotazovací tok neexistuje. Hľadá tok v tabuľke Active-Flow, s nastaveným príznakom `conversation`, ktorý obsahuje cieľovú adresu zhodnú so zdrojovou adresou spracovávaného paketu. Príznak `conversation` vyjadruje, že tok je dotazovací a môže byť k nemu vytvorený opačný tok - odpovedací. Ak sa jedná o IP protokol, porovnáva aj na základe cieľového a zdrojového portu. Ak zistí, že tok je odpoveďou k inému dátovému toku, vytvorí nový záznam v tabuľke Active-Flow. Pôjde síce o odpovedací tok, ale podľa koncepcie tohto kontroléra bude aj pre tento odpovedací tok vytvorený nový záznam v tabuľke Active-Flow. V tomto prípade nie je potrebné prepočítavať novú cestu ani kontrolovať pravidlá. Použije sa cesta z dotazovacieho toku, ktorej uzly sa invertujú. Odpovedací tok bude sieťovou topológiou smerovaný rovnakou trasou ako dotazovací tok. V prípade, že správca požaduje smerovanie odpovede inou trasou, je nevyhnutné definovať takúto komunikáciu prostredníctvom dvoch pravidiel typu `FLOW` v príslušnej politike.

Ak kontrolér nenájde k paketu dotazovací tok, nejedná sa o konverzáciu a je potrebné zistiť, či táto komunikácia je žiaduca, alebo bude zahodená. Kontrolér zistí, do ktorých dátových tokov, definovaných v zdrojovom kóde v jazyku OF-NCL tento paket patrí. Ak paket nepatrí do žiadneho dátového toku, je zahodený. V prípade, že paket vyhovuje niektorému deklarovanému dátovému toku v zdrojovom kóde, kontrolér zistí, podľa ktorého pravidla by mal s paketom zaobchádzať. Potrebuje nájst práve jedno pravidlo politiky, priradenej pro-

stredníctvom atribútu `CONTROL` v bloku špecifikácii OpenFlow switchu. Postupne prechádza pravidlá prvej a ďalšej politiky v poradí, v akom sú tieto zapísané v parametre atribútu `CONTROL`. Ak niektoré z pravidiel špecifikuje dátový tok, do ktorého paket patrí, je toto pravidlo zvolené za správne, prehľadávanie pravidiel sa považuje za úspešné a skončí. Na základe cesty `Path`, ktorá je definovaná týmto pravidlom, spočíta cestu, ktorou bude dátový tok smerovaný sieťovou topológiou k cieľovej stanici. Ak cesta nie je zvolená a je použitý zástupný symbol '`*`', prichádzajú do úvahy všetky možné cesty zo zdrojovej stanice k cieľovej stanici. Grafový algoritmus vyhľadá všetky dostupné cesty na základe popisu topológie v súbore so zdrojovým kódom. Viac o tomto prehľadávaní uvádzam v podkapitole 4.7.3. Ak blok špecifikácií tohto pravidla definuje algoritmus pre výber cesty pomocou atribútu `GRAPH`, vyberie sa jediná cesta z množiny všetkých dostupných ciest na základe zadaného algoritmu. Je možné implementovať aj vlastný algoritmus na výber cesty. Ak tento grafový algoritmus nie je špecifikovaný, vyberie sa cesta s najnižším ohodnotením. Pre neohodnotené linky to bude najkratšia cesta (keďže prednastavená cena linky je 1). Teraz je známa cesta, po ktorej bude paket doručený na cieľovú stanicu. Vygeneruje sa nový záznam do tabuľky Active-Flow o tomto dátovom toku. Zároveň kontrolér pridá nové pravidlo do Flow-table OpenFlow switcha, ktorý tento paket zaslal na analýzu.

4.7.3 Prehľadávanie topológie metódou DFS

Vyhľadávanie všetkých dostupných ciest je implementované algoritmom Depth-First Search. Tento algoritmus postupne rozgenerováva všetky uzly grafu, ktorú su tvorené deklarovanými OpenFlow switchmi. Začína v switchi, na ktorý bol prijatý paket. Pritom sleduje, aby nedošlo k rozgenerovaniu uzla, ktorý je jeho predchodcom. Týmto zabezpečí, že nedôjde k nájdeniu cesty, ktorá by tvorila cyklus. Algoritmus rozgenerováva iba uzly - OpenFlow switche, ktoré sú aktívne. Takisto sleduje, či linka, po ktorej je podľa deklarácie možné dostať sa do nasledujúceho uzla funguje a umožňuje prenos dát. Ďalší priestor na vylepšenie by bol v automatickom ohodnotení linky na základe vyťaženosti. Vyťaženosť by bola vyhodnocovaná napríklad na základe počtu dátových tokov, ktoré touto linkou prechádzajú, prípadne priamo podľa objemu prenášaných dát. Algoritmus prehľadávania do hĺbky bol implementovaný v [8], kde je aj detailnejšie popísaný.

4.7.4 Pseudo algoritmus riadenia OpenFlow siete

V tejto kapitole sa nachádza pseudo-algoritmus riadenia OpenFlow siete implementovaným kontrolérom. Tento algoritmus zachytáva najmä dôležité časti pri rozhodovaní kontroléra. Vstupom je paket, prijatý na OpenFlow kontrolér od switcha, ktorý pre tento paket nemá v internej Flow-Table definované žiadne pravidlo a teda nevie, ako má s paketom zaobchádzať. Kontrolér použije postupy, uvedené v predchádzajúcich kapitolách, a pokúsi sa nájsť správne pravidlo, podľa ktorého môže paket preposlať smerom k príjemcovi. Výstupom tohto algoritmu je súbor akcií, ktoré sa zapíšu ako pravidlo do Flow-Table príslušného OpenFlow switcha, ktorý inicioval spracovanie paketu. S týmito akciami je spoločne uložený aj tzv. `Match`. Je to súbor parametrov, špecifických pre daný dátový tok. Na všetky pakety, ktoré vyhovujú tomuto pravidlu je potom aplikovaný súbor akcií v ňom obsiahnutých.

```
def prijaty_novy_paket
  if paket_je_protokolu_arp {
    ARPManager( paket )
    ukonit_spracovanie()
  }
  # nasledujucu funkcionalitu implementuje metoda packetRoutine( ... )
  akcie = []
  if activeflow_obsahuje_zaznam_s_cielovou_mac_adresou_paketu {
  # sietovy tok je znamy a kontroler ma spocitanu jeho cestu
    tok = tabulka_activeflow[paket.cielova_mac_adresa]
    if cesta_pre_tok_obsahuje_aktualny_switch {
      if aktualny_switch_je_poslednym_na_ceste {
      # preposlanie paketu na koncove zariadenie - prijemcovi
        akcie << zmenit_cielovu_mac_adresu_na_adresu_prijemcu
        akcie << preposlat_cez_zadane_fyzicke_rozhranie
      } else {
      # preposlanie paketu na dalsi switch na ceste k prijemcovi
        akcie << preposlat_cez_zadane_fyzicke_rozhranie
      }
    } else {
      chyba()
    }
  } else {
  # sietovy tok nie je zatial znamy, kontroler najskor spocita cestu
    dotazovaci_tok = v_activeflow_najst_opacny_tok
    if dotazovaci_tok {
    # podarilo sa najst dotazovaci tok, aktualne spracovavany je odpovedaci
      novy_tok = pridat_aktualny_tok_do_activeflow
      novy_tok.cesta = dotazovaci_tok.cesta_v_invertovanej_forme
      akcie << zmenit_cielovu_mac_na_adresu_specificku_pre_tok
      akcie << preposlat_cez_zadane_fyzicke_rozhranie
    } else {
    # nepodarilo sa najst dotazovaci tok, bude spracovany ako samostatny
      datove_toky = zaradit_do_deklarovanych_datovych_tokov( paket )
      if nenajdeny_ziadny_datovy_tok {
      # pre tok z tohto paketu nie je zamyslana ziadna akcia-zahodit paket
        ukoncit_spracovanie()
      } else {
      # existuje aspon jeden deklarovany datovych tokov pre tento paket
        pre_kazdu_politiku_priradenu_aktualnemu_switchu {
          pre_kazde_pravidlo_tejto_politiky {
            if datove_toky_obsahuju_tok_specifikovany_v_tomto_pravidle {
            # bolo najdene pravidlo, ktore vyhovuje tomuto datovemu toku
              cesty = spocitaj_mozne_cesty( pravidlo )
              cesta = grafova_funkcia_specifikovana_pravidlom( cesty )
              vytvor_zaznam_v_tabulke_activeflow( paket, cesta )
              akcie << zmenit_cielovu_mac_na_adresu_specificku_pre_tok
              akcie << preposlat_cez_zadane_fyzicke_rozhranie
          }
        }
        }
        if nebolo_najdene_pravidlo_pre_tento_datovy_tok {
          ukonci_spracovanie()
  } } } }
  match = vytvorit_match_z_hlaviciek_paketu( paket )
  nastavit_tok_v_aktualnom_switchi( match, akcie )
  odoslat_paket_spracovany_kontrolerom( paket, akcie )
end
```

Kapitola 5

Testovanie a vyhodnotenie implementácie

Súčasťou tejto práce je aj vytvorenie testovacieho prostriedia pre implementovaný interpret jazyka OF-NCL. Topológia je navrhnutá s ohľadom na otestovanie a demonštrovanie rôznych situácií, ktoré môžu v počítačovej sieti nastať.

5.1 Testovacie prostredie MiniNet

OpenFlow je pomerne nová technológia. Vývoj softwaru v tomto prípade napreduje pred vývojom hardvérových zariadení. K testovaniu implementácie bolo preto použité virtuálne prostredie, vytvorené prostredníctvom nástroja MiniNet [4].
MiniNet je sieťový emulátor a dokáže vytvoriť realistickú virtuálnu počítačovú sieť. Táto pozostáva z OpenFlow switchov, koncových staníc a virtuálnych prepojov medzi nimi. Nástroj MiniNet je spustiteľný v prostredí operačného systému Linux. Odporúčaná a podporovaná je distribácia Linux Ubuntu verzie 12.04 a vyššie. MiniNet poskytuje vlastné API[1] v jazyku Python pre vytvorenie topológie virtuálnej počítačovej siete a nastavenie jej parametrov a umožňuje ovládať toto prostredie pomocou CLI[2].

Dôležité príkazy CLI v prostredí MiniNet sú:

net
: Zobrazí textový popis zapojenia virtuálnej sieťovej topológie.

nodes
: Zobrazí všetky existujúce uzly vo virtuálnej sieťovej topológii. Patria sem OpenFlow switche, kontroléry a pracovné stanice.

link (device1) (device2) down
: Príkaz umožní rozpojenie virtuálnej linky medzi dvoma zariadeniami. Zariadenia (device1) a (device2) sú odkazované symbolickými názvami, ktoré je možné zistiť pomocou predchádzajúcich príkazov. Tieto symbolické názvy sú nastavené v skripte pre vytvorenie MiniNet topológie. Opätovné zapnutie linky je možné zmenou posledného parametru tohto príkazu na `up`.

[1] Application Programming Interface. Je to protokol, ktorý umožňuje komunikáciu medzi softvérovými komponentami.

[2] Command-Line Interface. Poskytuje užívateľovi možnosť ovládať aplikáciu prostredníctvom príkazového riadku.

xterm (device1) (device2) ...
Tento príkaz otvorí terminálove okná pre všetky zadané zariadenia.

exit
Korektne rozpojí virtuálnu topológiu a ukončí nástroj MiniNet.

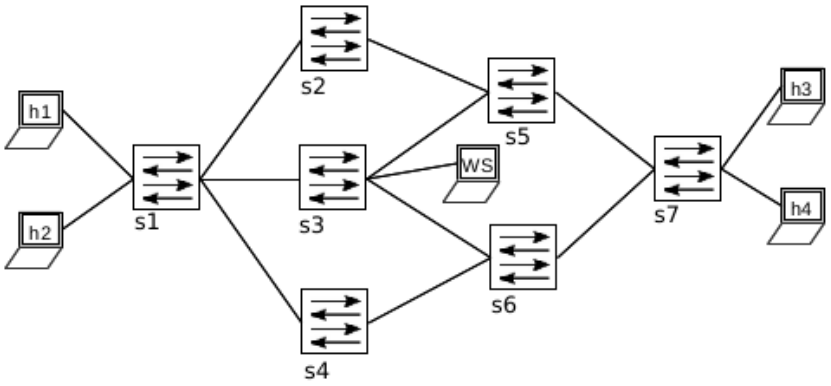
(device) (command)
Zadaním názvu zariadenia virtuálnej topológie ako prvého parametru je možné spúšťať ľubovolné príkazy na tomto zariadení, napríklad:

```
h1 ping h4
h1 ping 10.0.0.4
h4 ifconfig
h3 telnet 10.0.0.100 8080
```

5.2 Návrh topológie počítačovej siete

Topológia počítačovej siete navrhnutá pre testovacie a demonštračné účely je na Obrázku 5.1. Topológia sa skladá zo siedmych OpenFlow switchow `s1 - s7`. Celkovo sú v topológii zapojené štyri pracovné stanice a jeden webový server. Správca zamýšľa nasledujúcu konfiguráciu siete:

- Pracovné stanice môžu pristupovať k webovému serveru pomocou protokolov HTTP a HTTPS.
- Pracovné stanice môžu vzájomne testovať svoj stav pomocou nástroja Ping.
- Pracovné stanice nemôžu testovať stav webového servera pomocou nástroja Ping.
- Webový server môže testovať stav pracovných staníc pomocou nástroja Ping. Táto komunikácia musí prechádzať cez switch s7.
- Správca siete bude môcť pristupovať k webovému serveru pomocou protokolu SSH zo stanice h4.
- Akákoľvek iná komunikácia je na sieti zakázaná.



Obrázek 5.1: Zapojenie virtuálnej topológie pre demonštračné účely

5.3 Zapojenie sieťovej topológie

K zapojeniu virtuálnej sieťovej topológie je využitý nástroj MiniNet, ktorý bol predstavený na začiatku tejto kapitoly. Sieťová topológia je popísaná v konfiguračnom súbore `topo.py`. Využíva k tomu prostriedky knižnice MiniNet. Na základe tohto skriptu vygeneruje MiniNet sieťovú topológiu.

Názov	**IP adresa**	**MAC adresa**	**Switch**	**Popis**
h1	10.0.0.1	12:80:d6:08:10:df	s1	Pracovná stanica
h2	10.0.0.2	d6:df:3b:f9:e4:ae	s1	Pracovná stanica
h3	10.0.0.3	26:bb:e9:42:2b:12	s7	Pracovná stanica
h4	10.0.0.4	f2:7e:a2:05:33:ee	s7	Pracovná stanica
ws	10.0.0.100	8e:bb:77:9d:9f:c6	s3	Webový server

Tabulka 5.1: Konfigurácia koncových zariadení

```
from mininet.topo import Topo

class BPdemo( Topo ):
  def __init__(self):
    Topo.__init__( self )
    s1 = self.addSwitch('s1')
    s2 = self.addSwitch('s2')
    s3 = self.addSwitch('s3')
    s4 = self.addSwitch('s4')
    s5 = self.addSwitch('s5')
    s6 = self.addSwitch('s6')
    s7 = self.addSwitch('s7')

    h1 = self.addHost('h1', ip='10.0.0.1', mac='12:80:d6:08:10:df')
    self.addLink(h1,s1)
    h2 = self.addHost('h2', ip='10.0.0.2', mac='d6:df:3b:f9:e4:ae')
    self.addLink(h2,s1)
    h3 = self.addHost('h3', ip='10.0.0.3', mac='26:bb:e9:42:2b:12')
    self.addLink(h3,s7)
    h4 = self.addHost('h4', ip='10.0.0.4', mac='f2:7e:a2:05:33:ee')
    self.addLink(h4,s7)

    ws = self.addHost('ws', ip='10.0.0.100', mac='8e:bb:77:9d:9f:c6')
    self.addLink(ws,s3)

    self.addLink(s1,s2)
    self.addLink(s1,s3)
    self.addLink(s1,s4)

    self.addLink(s2,s5)
    self.addLink(s3,s5)
    self.addLink(s3,s6)
```

```
    self.addLink(s4,s6)

    self.addLink(s5,s7)
    self.addLink(s6,s7)

topos = { 'bpdemo': ( lambda: BPdemo() ) }
```

Skript pre spustenie virtualizovaného prostredia je pripravený.

5.4 Zostavenie programu v jazyku OF-NCL

Teraz je potrebné popísať zamýšľanú konfiguráciu OpenFlow siete v jazyku OF-NCL. Najskôr je vhodné deklarovať všetky OpenFlow switche (s1 - s7), ktoré sa v topológii vyskytujú a budú ovládané kontrolérom. V deklarácii každého switcha sa vyskytuje atribút `DATAPATH`. Je to identifikátor OpenFlow switchu, pridelený nástrojom MiniNet. Ten ho pridelí na základe názvu, takže pre switch `s1` bude mať hodnotu '1', atď. Následne sú zadefinované rozhrania typu `INTERFACE` a `INTERCONN` (vysvetlené v kapitole 3.1.5) na každom switchi. Za symbolickým názvom rozhrania nasleduje znak ':' a číslo alebo rozsah fyzických portov, ktoré sú k tomuto rozhraniu priradené.

```
SWITCH s1 {
  DATAPATH 1
  INTERFACE pc:1-2
  INTERCONN interconnect_s2:3
  INTERCONN interconnect_s3:4
  INTERCONN interconnect_s4:5
};

SWITCH s2 {
  DATAPATH 2
  INTERCONN interconnect_s1:1
  INTERCONN interconnect_s5:2
};

SWITCH s3 {
  DATAPATH 3
  INTERFACE pc:1
  INTERCONN interconnect_s1:2
  INTERCONN interconnect_s5:3
  INTERCONN interconnect_s6:4
};

SWITCH s4 {
  DATAPATH 4
  INTERCONN interconnect_s1:1
  INTERCONN interconnect_s6:2
};
```

```
SWITCH s5 {
  DATAPATH 5
  INTERCONN interconnect_s2:1
  INTERCONN interconnect_s3:2
  INTERCONN interconnect_s7:3
};

SWITCH s6 {
  DATAPATH 6
  INTERCONN interconnect_s3:1
  INTERCONN interconnect_s4:2
  INTERCONN interconnect_s7:3
};

SWITCH s7 {
  DATAPATH 7
  INTERFACE pc:1-2
  INTERCONN interconnect_s5:3
  INTERCONN interconnect_s6:4
};
```

Teraz sú deklarované všetky OpenFlow switche. Ďalej je potrebné deklarovať linky, ktorými sú OpenFlow switché prepojené (podľa obrázka 5.1).

```
LINK s1_s2 BETWEEN s1.interconnect_s2 AND s2.interconnect_s1;
LINK s1_s3 BETWEEN s1.interconnect_s3 AND s3.interconnect_s1;
LINK s1_s4 BETWEEN s1.interconnect_s4 AND s4.interconnect_s1;

LINK s2_s5 BETWEEN s2.interconnect_s5 AND s5.interconnect_s2;
LINK s3_s5 BETWEEN s3.interconnect_s5 AND s5.interconnect_s3;
LINK s3_s6 BETWEEN s3.interconnect_s6 AND s6.interconnect_s3;
LINK s4_s6 BETWEEN s4.interconnect_s6 AND s6.interconnect_s4;

LINK s5_s7 BETWEEN s5.interconnect_s7 AND s7.interconnect_s5;
LINK s6_s7 BETWEEN s6.interconnect_s7 AND s7.interconnect_s6;
```

Týmto je demonštračná virtuálna topológia popísaná. Pomocou atribútu `COST` je možné pri každej linke nastaviť jej cenu. Táto sa potom môže uplatniť pri výbere trasy pre dátový tok, pokiaľ sa v politike aplikuje niektorá grafová funkcia. Teraz budú postupne nadefinované požiadavky (vyššie špecifikované) na chovanie siete.

5.5 Definícia chovania siete

Chovanie siete je definované pomocou pravidiel zapísaných v bloku Policy. Každé pravidlo sa skladá zo špecifikácie dátového toku, popisu zamýšľanej cesty cez sieťovú topológiu a rozširujúcich atribútov. Pravidlá môžu byť typu `FLOW` alebo `CONVERSATION`. Tieto pravidlá sú bližšie vysvetlené v časti 3.1.9.

Najskôr je nutné popísať zamýšľané dáta, resp. dátové toky, ktoré budú interpretom riadené. Špecifikácia dátového toku pre prvé zamýšľané pravidlo bude vyzerať nasledovne.

"Pracovné stanice môžu pristupovať k webovému serveru pomocou protokolov HTTP a HTTPS."

```
#define HTTP 80
#define HTTPS 443

FLOW webserver_access {
  SRC.IP = 10.0.0.1 - 10.0.0.4
  DST.IP = 10.0.0.100
  DST.PN = HTTP, HTTPS
  PT = TCP
};
```

Najskôr boli kvôli sprehľadneniu zadefinované čísla portov 80 a 443, ktoré sú štandardne využívané protokolmi HTTP a HTTPS. Nasleduje deklarácia dátového toku, ktorý bol symbolicky pomenovaný názvom `webserver_access`. V tele tejto deklarácie špecifikuje atribút `SRC.IP` rozsah IP adries pracovných staníc v demonštračnej topológii. IP adresa atribútu `DST.IP` patrí webovému serveru a čísla portov, vyjadrené atribútom `DST.PN` sú pomocou predchádzajúcich definícií nastavené na hodnotu 80 alebo 443. Atribút `PT` hovorí, že komunikácia môže prebiehať iba pomocou protokolu TCP.

Analogicky budú zadefinované aj ostatné dátové toky, ktoré je potrebné zachytiť pre aplikovanie pravidiel.

"Pracovné stanice môžu vzájomne testovať svoj stav pomocou nástroja Ping."

```
FLOW hosts_ping {
  SRC.IP = 10.0.0.1 - 10.0.0.4
  DST.IP = 10.0.0.1 - 10.0.0.4
  PT = ICMP
};
```

"Pracovné stanice nemôžu testovať stav webového servera pomocou nástroja Ping."

Pre realizáciu tejto požiadavky nie je potrebné aplikovať žiadne pravidlo. Dostatočné je, že neexistuje žiadne pravidlo, ktoré by explicitne túto komunikáciu umožňovalo.

"Webový server môže testovať stav pracovných staníc pomocou nástroja Ping. Táto komunikácia musí prechádzať cez switch s7."

```
FLOW ws_pingall {
  SRC.IP = 10.0.0.100
  DST.IP = 10.0.0.1 - 10.0.0.4
  PT = ICMP
};
```

```
PATH thr_s7 {
  WAYPOINT s7
};
```

Požiadavka, že komunikácia musí prechádzať cez OpenFlow switch s7 je vyjadrená prostredníctvom entity Path. Atribút `WAYPOINT` selektuje iba cesty, ktoré obsahujú uzol `s7`.

"Správca siete bude môcť pristupovať k webovému serveru pomocou protokolu SSH zo stanice h4."

```
#define SSH 22

FLOW h4_ssh_ws {
  SRC.IP = 10.0.0.4
  DST.IP = 10.0.0.100
  DST.PN = SSH
  PT = TCP
};
```

"Na sieti je akákoľvek iná komunikácia zakázaná."

Táto požiadavka je vyriešená, keďže interpret akceptuje iba komunikáciu, ktorá je explicitne povolená.

Posledným krokom je zapísať politiky v jazyku OF-NCL. Politiky boli vysvetlené v kapitole 3.1.9 a pre demonštračné účely budú vytvorené dve samostatné politiky. Prvá s názvom `global` bude aplikovaná na všetky OpenFlow zariadenia, ktoré spracovávajú komunikáciu od pripojených koncových zariadení. Sú to OpenFlow switche `s1`, `s3` a `s7`.

```
POLICY global {
  CONVERSATION webserver_access PATH * {
    GRAPH shortest
  };
  CONVERSATION h4_ssh_ws PATH * {
    GRAPH shortest
  };
  CONVERSATION ws_pingall PATH thr_s7 {
    GRAPH random
    TIMEOUT-HARD 30s
  };
};
```

Konverzácia pre dátový tok popísaný entitou `ws_pingall` bude smerovaná cez OpenFlow switch `s7` na základe deklarovanej cesty `thr_s7`. Grafová funkcia `random` vyberie ľubovolnú cestu z množiny možných ciest. Dĺžka tohto toku je pevne stanovaná na 30 sekúnd. Po tomto čase tok expiruje a bude ustanovený nový tok, ktorý môže využívať inú cestu.

Druhá politika nazvaná `ping` bude aplikovaná iba na OpenFlow switche `s1` a `s7`, ku ktorým sú pripojené pracovné stanice. Táto politika ošetruje "pingovanie"medzi pracovnými stanicami navzájom. Na iných OpenFlow switchoch by nemala žiadne opodstatnenie.

```
POLICY ping {
  CONVERSATION hosts_ping PATH * {
    GRAPH random
    TIMEOUT-HARD 30s
  };
};
```

Teraz je potrebné do deklarácií OpenFlow switchov pridať pomocou atribútu `CONTROL` informáciu o politikách, ktoré bude switch akceptovať. OpenFlow switche `s1` a `s7` budú rozšírené o atribút `CONTROL global, ping` a switch `s3` o atribút `CONTROL global`. Vo výsledku bude ich deklarácia vyzerať nasledovne:

```
SWITCH s1 {
  DATAPATH 1
  INTERFACE pc:1-2
  INTERCONN interconnect_s2:3
  INTERCONN interconnect_s3:4
  INTERCONN interconnect_s4:5
  CONTROL global, ping
};

SWITCH s3 {
  DATAPATH 3
  INTERFACE pc:1
  INTERCONN interconnect_s1:2
  INTERCONN interconnect_s5:3
  INTERCONN interconnect_s6:4
  CONTROL global
};

SWITCH s7 {
  DATAPATH 7
  INTERFACE pc:1-2
  INTERCONN interconnect_s5:3
  INTERCONN interconnect_s6:4
  CONTROL global, ping
};
```

5.6 Spúšťací skript netctrl

K spusteniu interpretu jazyka OF-NCL spolu s OpenFlow kontrolérom Trema bol vytvorený skript v jazyku Bash. Pri jeho spustení sa vyžaduje zadanie minimálne jedného parametru príkazového riadku pomocou prepínača `-c`, prostredníctvom ktorého sa zadáva cesta k súboru so zdrojovým programom v jazyku OF-NCL.

Skript `netctrl` podporuje zadanie nasledujúcich parametrov:

-c Cesta k súboru so zdrojovým kódom v jazyku OF-NCL

-p Číslo TCP portu, na ktorom bude OpenFlow kontrolér spustený

-k Ukončiť všetky spustené Trema kontroléry

-d Spustiť OpenFlow kontrolér v režime démona

-o Presmerovanie štandardného výstupu do súboru

-e Presmerovanie štandardného chybového výstupu do súboru

Skript sa ukončí po prijatí signálu SIGINT. Tento môže byť v operačných systémoch Linux zaslaný napríklad stlačením klávesovej kombinácie `CTRL+c`.

Nasledujúce príklady ukazujú možné spustenie vlastného OpenFlow kontroléra pomocou skriptu netctrl.

Spustenie OpenFlow kontroléra so zdrojovým súborom `script.ofncl` na štandardnom porte `6633`.

```
user@localhost:~/ netctrl -c script.ofncl
```

Spustenie OpenFlow kontroléra so zdrojovým súborom `script.ofncl` na zvolenom porte `52000`.

```
user@localhost:~/ netctrl -c script.ofncl -p 52000
```

Spustenie OpenFlow kontroléra so zdrojovým súborom `script.ofncl` v režime démona na zvolenom porte `52000`. Výstupy budú zahodené.

```
user@localhost:~/ netctrl -c script.ofncl -d -p 52000
```

Ukončenie všetkých spustených OpenFlow kontrolérov včítane tých, ktoré bežia v režime démona.

```
user@localhost:~/ netctrl -k
```

Spustenie OpenFlow kontroléra so zdrojovým súborom `script.ofncl` v režime démona na zvolenom porte `52000`. Výstup `STDOUT` bude presmerovaný do súboru `std.out` a `STDERR` do súboru `std.err`.

```
user@localhost:~/ netctrl -c script.ofncl -d -o std.out -e std.err
```

5.7 Spustenie sieťovej topológie a OpenFlow kontroléra

Teraz je zostavený celý program pre zamýšľanú demonštračnú topológiu a jej chovanie v jazyku OF-NCL. Nasleduje spustenie interpretu a OpenFlow kontroléra. K tomu slúži nástroj `netctrl` predstavený v podkapitole 5.6. Spustenie interpretu a následne kontroléra na porte `6633` je možné pomocou príkazu:

```
user@localhost:~$ netctrl -c /home/user/openflow/bpdemo.ofncl -p 6633
```

Interpret jazyka OF-NCL spracuje vstupný súbor, ktorý je zadaný prostredníctvom prepínača `-c`. Potom sa spustí program, ktorý bude čakať na pripojenie OpenFlow switchov a následne ich bude riadiť.

Nasleduje spustenie virtuálnej topológie cez nástroj MiniNet. Tento nástroj je nutné spúšťať s oprávnením superužívateľa. K tomu je možné využiť príkaz `sudo`.

```
user@localhost:~$ sudo mn --controller=remote,ip=127.0.0.1,port=6633
--custom /home/user/mininet/topo.py --topo bptopo
```

Vyššie uvedený príkaz vytvorí virtuálnu demonštračnú topológiu. Je nastavené využitie externého kontroléra prepínačom `--controller`, keďže MiniNet obsahuje aj zabudovaný vlastný. OpenFlow switche, spustené prostredníctvom MiniNet-u, sa teraz budú pripájať k lokálne spustenému kontroléru na TCP porte 6633. Prepínač `--custom` zvolí súbor, ktorý obsahuje skript pre vygenerovanie virtuálnej topológie a prepínač `--topo` vyberá konkrétnu topológiu v tomto súbore.
Pri problémoch s nástrojom MiniNet je možné použiť "čistiacu"funkciu tohto nástroja príkazom:

```
user@localhost:~$ sudo mn -c
```

Túto funkciu je vhodné spustiť pred každým vygenerovaním novej topológie. Možno tým predísť prípadným problémom pri jej vytváraní.
Teraz je virtuálna topológia spustená. Pomocou CLI nástroja MiniNet mu možno zadávať riadiace príkazy, prípadne spúšťať nástroje priamo na virtuálnych koncových staniciach. V demonštračnej topológii existujú pracovné stanice `h1 - h4` a webový server `ws`. Nasledujúci príkaz otvorí nové terminálové okná pracovných staníc a webového servera:

```
mininet> xterm h1 h2 h3 h4 ws
```

Otestovať požadovanú funkčnosť je možné jednoducho z terminálového okna príslušnej pracovnej stanice. Na obrázkoch v prílohe D sú zachytené terminálové okná jednotlivých staníc, ktoré ukazujú funkčnosť zamýšľanej konfigurácie demonštračnej sieťovej topológie.

Z uvedeného je zrejmé, že "pingovanie"navzájom medzi pracovnými stanicami funguje. Ďalšie obrázky v tejto prílohe ukazujú spustený webový server, ktorý poskytuje svoje služby na porte 80 pre pracovné stanice v sieti a službu SSH pre pracovnú stanicu `h4`.

Takto som vo svojej práci overil funkčnosť virtuálnej demonštračnej siete. Všetky požiadavky, ktoré boli špecifikované a zapísané v jazyku OF-NCL demonštračná sieť spĺňa. Bolo teda preukázané, že interpret dokáže na základe zápisu v jazyku OF-NCL spracovať vstupný program, riadiť OpenFlow switche a teda spravovať sieťovú topológiu.

Kapitola 6

Záver

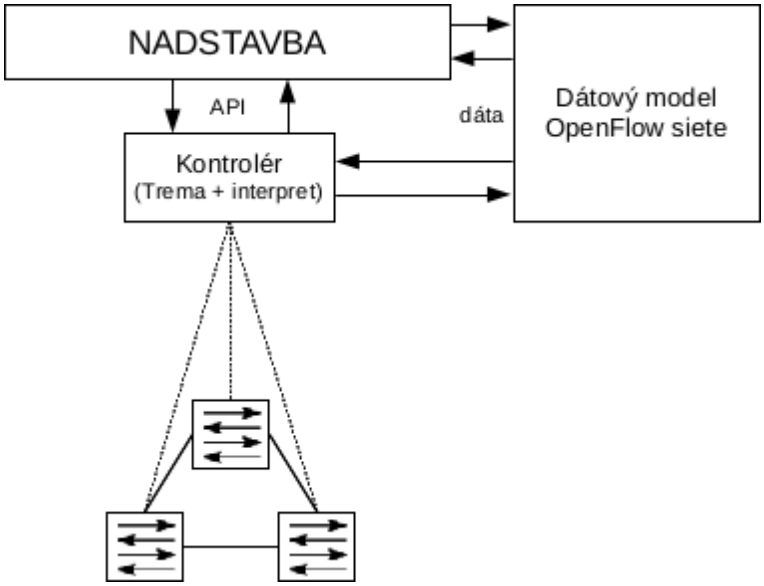
Vo svojej bakalárskej práci som sa venoval návrhu jazyka pre popis a riadenie počítačových sietí pomocou technológie OpenFlow. Jazyk OpenFlow-Network Control Language, vytvorený v tejto bakalárskej práci, umožňuje správcom počítačovej siete vytvoriť program, ktorý bude riadiť OpenFlow sieť na základe politík opísaných v kapitole 3.1.9. V tomto jazyku je možné špecifikovať dátové toky, ktoré sú definované na základe parametrov paketov. Pre tieto dátové toky následne umožňuje stanoviť smerovanie zamýšľanou sieťovou topológiou. Vytvoriť charakteristiku sieťovej topológie je možné pomocou deklarácie sieťových zariadení OpenFlow a prepojov medzi nimi. Vo svojej práci predkladám čitateľovi spôsob implementácie interpretu pre navrhnutý jazyk OF-NCL.

Interpret na základe programu v jazku OF-NCL riadi OpenFlow sieť. Generuje pravidlá pre smerovanie dátových tokov. Tieto pravidlá pridáva a spravuje v interných tabuľkách OpenFlow zariadení. Udržiava si svoj vnútorný model počítačovej siete, aby bol schopný bezodkladne reagovať na novú situáciu, ktorá v OpenFlow sieti môže vzniknúť. Takisto sa snaží o predpočítanie niektorých údajov následne potrebných. Aj keď z hľadiska časovej náročnosti existuje stále priestor na zlepšenie, hlavným cieľom bolo demonštrovať možnosť nasadiť novú technológiu OpenFlow v bežne známych situáciách.

Sila OpenFlow technológie je najmä v zavedení arbitra do počítačových sietí. Tým je možné centralizovať ich správu. Moje riešenie poskytuje pomerne kvalitný dohľad aj nad rozsiahlou počítačovou sieťou. Arbiter má prehľad o celej sieti a má možnosť promptne zasahovať do jej konfigurácie.

Ako autor práce mám záujem pokračovať vo vývoji tohto interpretu. Vidím v technológii OpenFlow nové možnosti a prínosy, ktoré pomôžu rozšíriť a sprehľadniť správu počítačových sietí. Rozvoju tejto technológie by výrazne napomohol aj rýchlejší vývoj hardwarových zariadení, ktoré by ju podporovali.

Pri návrhu a implementácii interpretu, vytvoreného v tejto práci, bol kladený dôraz na jeho budúcu rozšíriteľnosť. Tento interpret ponúka základný prostriedok pre riadenie OpenFlow počítačovej siete. Aby bolo možné naplno využiť výhody a prostriedky OpenFlow technológie, je nevyhnutné rozširovať jeho funkcionalitu. To je možné postupným dopĺňaním programových komponent a rozširovaním existujúcich. Práve existencia dátového modelu umožňuje programátorovi zjednodušenú prácu pri tvorbe a údržbe interpretu.



Obrázek 6.1: Nadstavba interpretu jazyka OF-NCL

Ďaľšou možnosťou rozširovania interpretu je vytvorenie novej programovej vrstvy. Táto by pomocou API rozhrania komunikovala priamo s interpretom a pristupovala aj do jeho dátového modelu, ako je znázornené na obrázku 6.1. Ošetrovala by prípadné neočakávané alebo kritické situácie vzniknuté v sieti. Týmto by odľahčila interpret a umožňila mu sústrediť sa na jeho kľúčové problémy, ktorými je riešiť bežnú prevádzku v sieti na základe programu zamýšľaného správcom siete.

V tejto práci som definoval jazyk OpenFlow Network Control Language. Následne som implementoval interpret pre tento jazyk, ktorý vygeneruje dátový model siete. Súčasťou implementácie je aj OpenFlow kontrolér. Ten na základe dátového modelu riadi OpenFlow sieť. Dátový model sa dopĺňa a modifikuje podľa aktuálneho stavu siete. Táto implementácia bola demonštrovaná vo virtuálnom prostredí, vytvoreného nástrojom MiniNet. Aplikácia je rozšíriteľná a umožňuje zahrnutie nových programových komponent.

Literatura

[1] Big Switch Networks: The Leader in Open Software Defined Networking.
URL http://www.bigswitch.com/

[2] Documentation for Trema.
URL http://rubydoc.info/github/trema/trema/master/frames

[3] Trema: Full-Stack OpenFlow Framework in Ruby and C.
URL http://trema.github.io/trema/

[4] Mininet: An Instant Virtual Network on your Laptop (or other PC). [online], 2013.
URL http://mininet.org/

[5] Meduna, A.;Lukáš, R.: Studijní opora: Formálni jazyky a překladače. 2012.

[6] Open Networking Foundation: Software-Defined Networking: The New Norm for Networks. [online], 2012-04-13 [cit. 2013-05-01].
URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[7] Open Networking Foundation: OpenFlow Switch Specification. [online], 2012-09-06 [cit. 2013-05-01].
URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf

[8] Zbořil, F.;Zbořil, F.: Studijní opora: Základy umělé inteligence. 2006.

Příloha A

Gramatika jazyka OF-NCL

```
<begin> → <device> <next_device>
<next_device> → <device> <next_device>
<next_device> → EPSILON
<device> → <switch>
<device> → <link>
<device> → <path>
<device> → <flow>
<device> → <policy>

<switch> → SWITCH <switch_name> <specs>;
<specs> → EPSILON
<specs> → { <specs_line> }
<specs_line> → <attribute> \n <specs_line>
<specs_line> → EPSILON
<attribute> → DATAPATH <datapath_id>
<attribute> → INTERFACE <interface_name> : <port_range>
<port_range> → <start> <upto>
<upto> → EPSILON
<upto> → - <end_port>
<attribute> → INTERCONN <interface_name> : <port_number>
<attribute> → CONTROL <policy_name> <policies>
<policies> → , <policy_name> <policies>
<policies> → EPSILON

<link> → LINK <link_name> BETWEEN <device_name>.<device_interconn> AND
<device_name>.<device_interconn> <specs>;
<attribute> → COST = <cost_value>

<path> → PATH <path_name> <specs>;
<attribute> → WAYPOINT <wp_device> <wps>
<wps> → EPSILON
<wps> → , <wp_device> <wps>


<flow> → FLOW <flow_name> <specs>;
```

```
<attribute> → SRC.IP = <ip_start> <ip_range>
<ip_range> → EPSILON
<ip_range> → - <ip_end>
<attribute> → SRC.PN = <pn_start> <pn_range>
<pn_range> → EPSILON
<pn_range> → - <pn_end>
<attribute> → DST.IP = <ip_start> <ip_range>
<attribute> → DST.PN = <pn_start> <pn_range>
<attribute> → PT = <protocol> <pts>
<pts> → , <protocol> <pts>
<pts> → EPSILON

<policy> → POLICY <policy_name> <specs>;
<attribute> → <rule_type> <flow_name> PATH <path_name> <rule_specs>
<rule_type> → FLOW
<rule_type> → CONVERSATION
<rule_specs> → EPSILON
<rule_specs> → { <rule_specs_line> };
<rule_specs_line> → EPSILON
<rule_specs_line> → <attribute> \n <rule_specs_line>
<attribute> → GRAPH <graph_name>
<attribute> → TIMEOUT-HARD <time>
<attribute> → TIMEOUT-IDLE <time>
```

Příloha B

Kľúčové slová jazyka OpenFlow Network Control Language

```
SWITCH
LINK
PATH
FLOW
POLICY
BETWEEN
AND
INTERFACE
INTERCONN
DATAPATH
SRC
DST
PT
CONTROL
WAYPOINT
DESTINATION
CONVERSATION
GRAPH
COST
TIMEOUT
```

Príloha C

Zdrojový kód demonštračnej topológie v jazyku OF-NCL

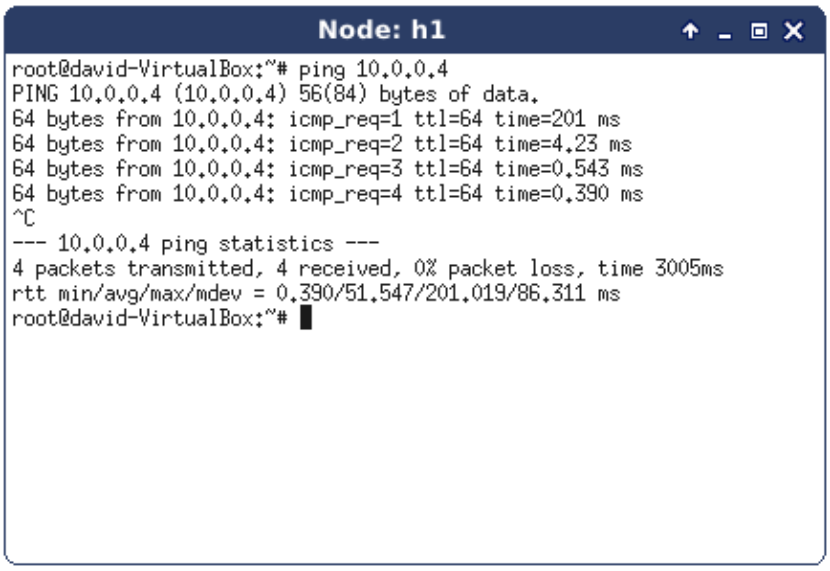
```
SWITCH s1 {
  DATAPATH 1
  INTERFACE pc:1-2
  INTERCONN interconnect_s2:3
  INTERCONN interconnect_s3:4
  INTERCONN interconnect_s4:5
  CONTROL global, ping
};

SWITCH s2 {
  DATAPATH 2
  INTERCONN interconnect_s1:1
  INTERCONN interconnect_s5:2
};

SWITCH s3 {
  DATAPATH 3
  INTERFACE pc:1
  INTERCONN interconnect_s1:2
  INTERCONN interconnect_s5:3
  INTERCONN interconnect_s6:4
  CONTROL  global
};

SWITCH s4 {
  DATAPATH 4
  INTERCONN interconnect_s1:1
  INTERCONN interconnect_s6:2
};

SWITCH s5 {
  DATAPATH 5
```

```
  INTERCONN interconnect_s2:1
  INTERCONN interconnect_s3:2
  INTERCONN interconnect_s7:3
};

SWITCH s6 {
  DATAPATH 6
  INTERCONN interconnect_s3:1
  INTERCONN interconnect_s4:2
  INTERCONN interconnect_s7:3
};

SWITCH s7 {
  DATAPATH 7
  INTERFACE pc:1-2
  INTERCONN interconnect_s5:3
  INTERCONN interconnect_s6:4
  CONTROL global, ping
};

LINK s1_s2 BETWEEN s1.interconnect_s2 AND s2.interconnect_s1;
LINK s1_s3 BETWEEN s1.interconnect_s3 AND s3.interconnect_s1;
LINK s1_s4 BETWEEN s1.interconnect_s4 AND s4.interconnect_s1;

LINK s2_s5 BETWEEN s2.interconnect_s5 AND s5.interconnect_s2;
LINK s3_s5 BETWEEN s3.interconnect_s5 AND s5.interconnect_s3;
LINK s3_s6 BETWEEN s3.interconnect_s6 AND s6.interconnect_s3;
LINK s4_s6 BETWEEN s4.interconnect_s6 AND s6.interconnect_s4;

LINK s5_s7 BETWEEN s5.interconnect_s7 AND s7.interconnect_s5;
LINK s6_s7 BETWEEN s6.interconnect_s7 AND s7.interconnect_s6;

FLOW webserver_access {
  SRC.IP = 10.0.0.1 - 10.0.0.4
  DST.IP = 10.0.0.100
  DST.PN = 80, 443
  PT = TCP
};

FLOW h4_ssh_ws {
  SRC.IP = 10.0.0.4
  DST.IP = 10.0.0.100
  PT = TCP
  DST.PN = 22
};

FLOW ws_pingall {
  SRC.IP = 10.0.0.100
```

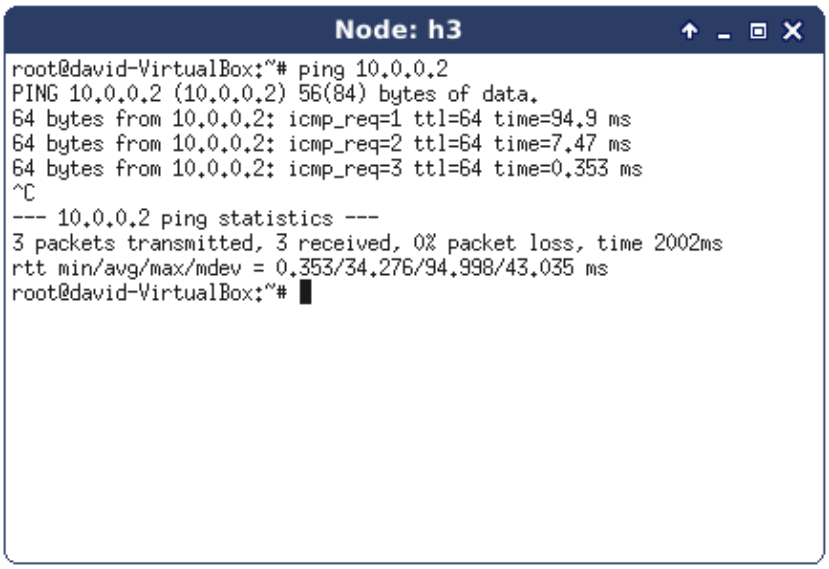
```
  DST.IP = 10.0.0.1 - 10.0.0.4
  PT = ICMP
};

PATH thr_s7 {
  WAYPOINT s7
};

POLICY global {
  CONVERSATION webserver_access PATH * {
    GRAPH shortest
  };
  CONVERSATION h4_ssh_ws PATH * {
    GRAPH shortest
  };
  CONVERSATION ws_pingall PATH thr_s7 {
    GRAPH random
    TIMEOUT-HARD 30s
  };
};

FLOW hosts_ping {
  SRC.IP = 10.0.0.1 - 10.0.0.4
  DST.IP = 10.0.0.1 - 10.0.0.4
  PT = ICMP
};

POLICY ping {
  CONVERSATION hosts_ping PATH * {
    GRAPH random
    TIMEOUT-HARD 30s
  };
};
```

Příloha D

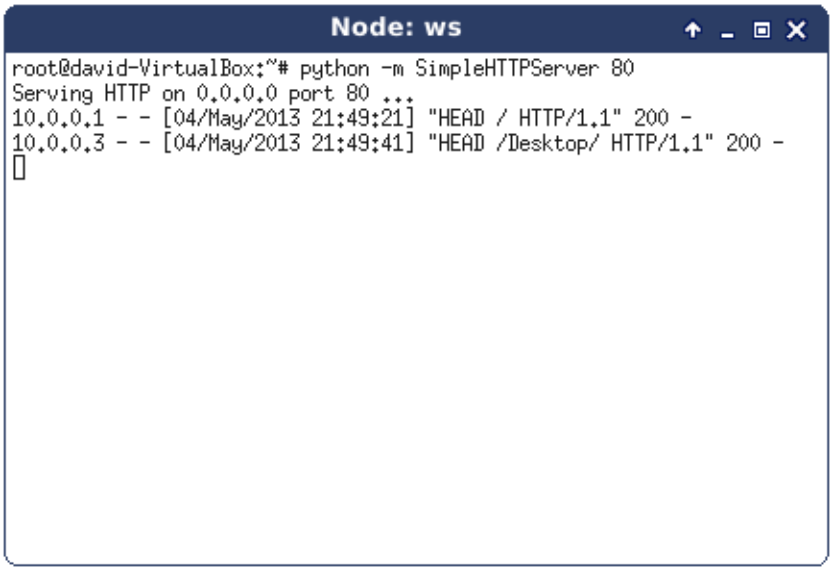
Výstupy z testovania demonštračnej topológie



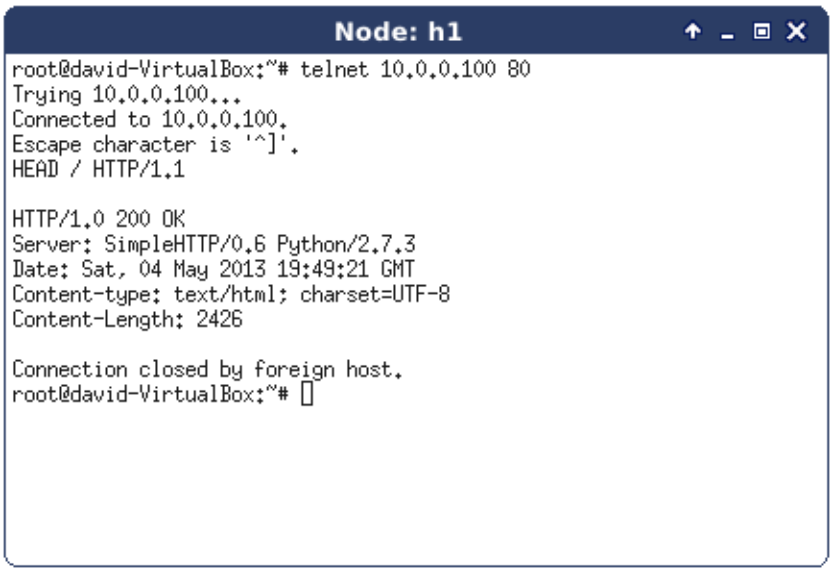
Obrázek D.1: Pracovná stanica h1 pinguje stanicu h4.



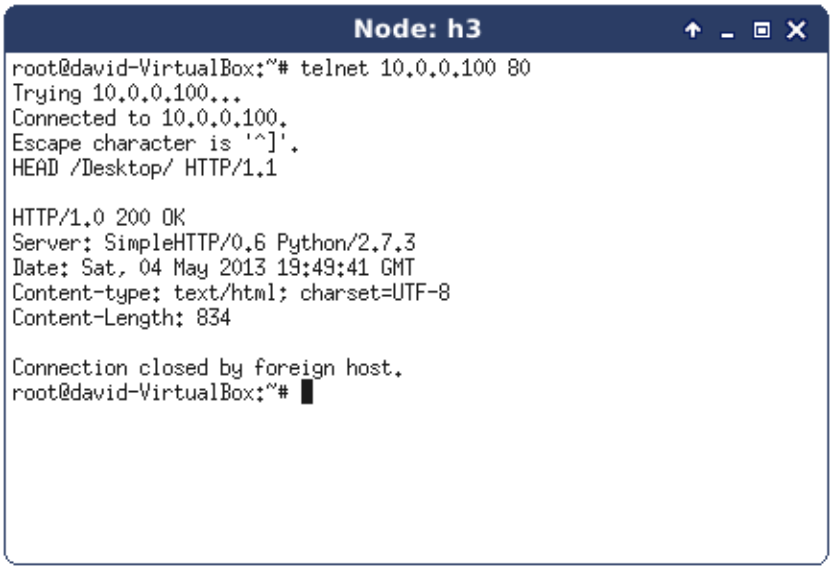
Obrázek D.2: Pracovná stanica h3 pinguje stanicu h2.



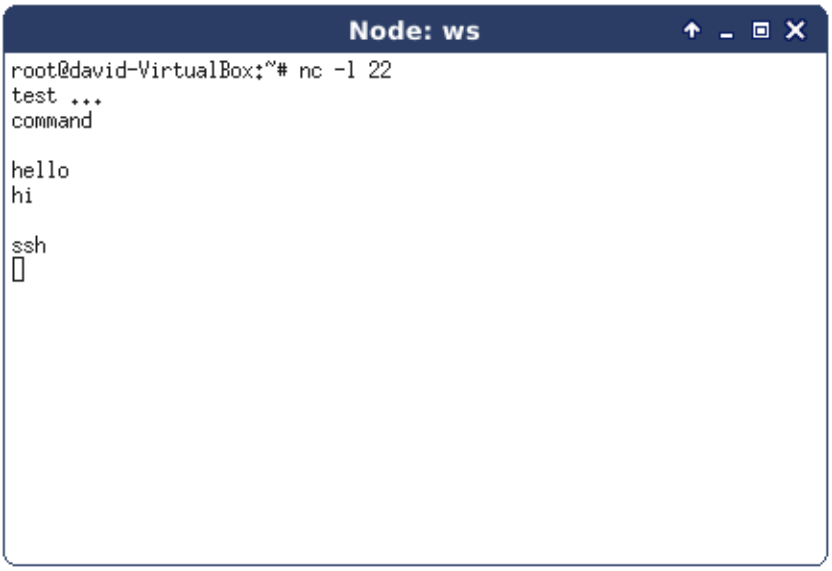
Obrázek D.3: Webový server SimpleHTTPServer spustený na webovom serveri.



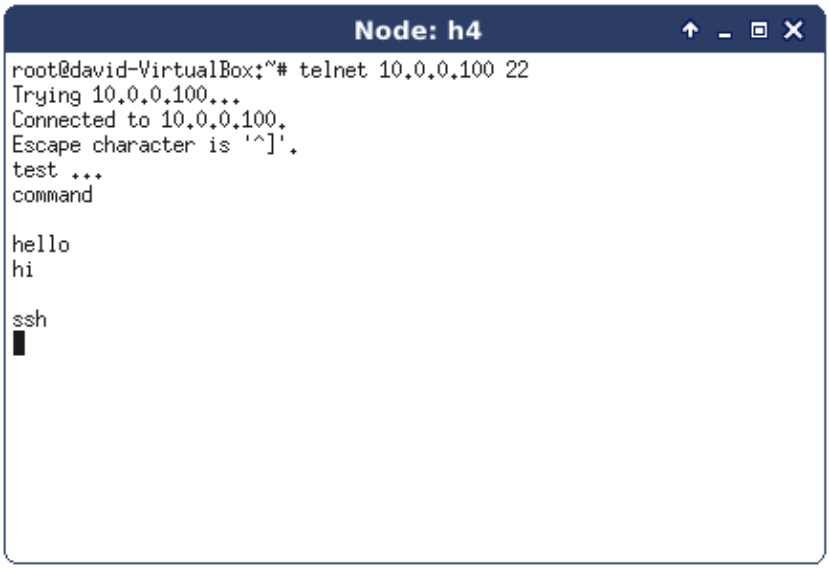
Obrázek D.4: Pracovná stanica h1 komunikuje s webovým serverom.



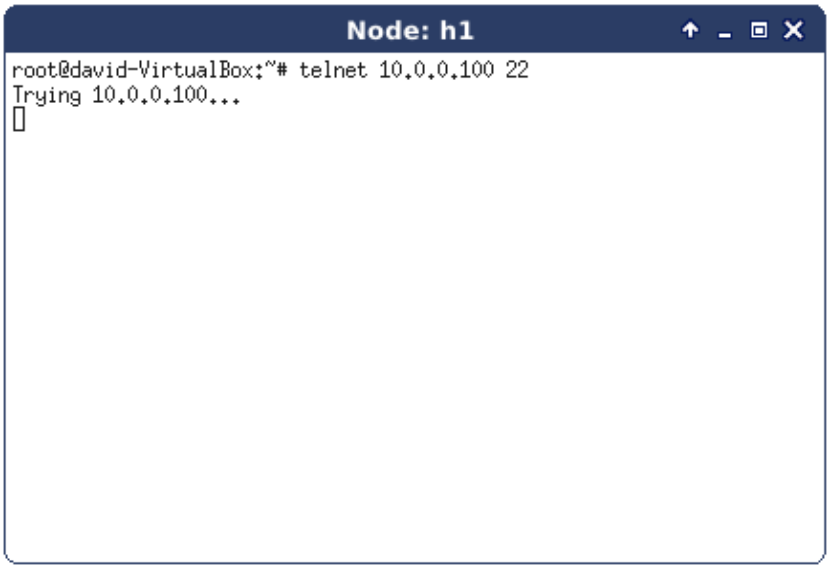
Obrázek D.5: Pracovná stanica h3 komunikuje s webovým serverom.



Obrázek D.6: Webový server načúva na porte 22.



Obrázek D.7: Pracovná stanica h4 otvorila spojenie s webovým serverom na porte 22.

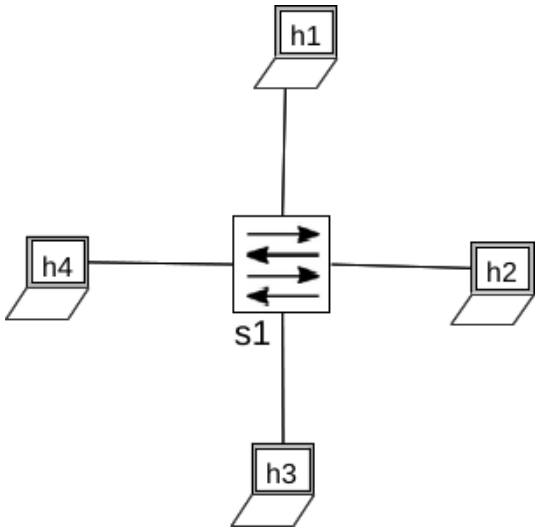


Obrázek D.8: Pracovná stanica h1 nedokázala vytvoriť spojenie s webovým serverom na porte 22, pretože podľa zamýšľanej konfigurácie na to nemá oprávnenie.

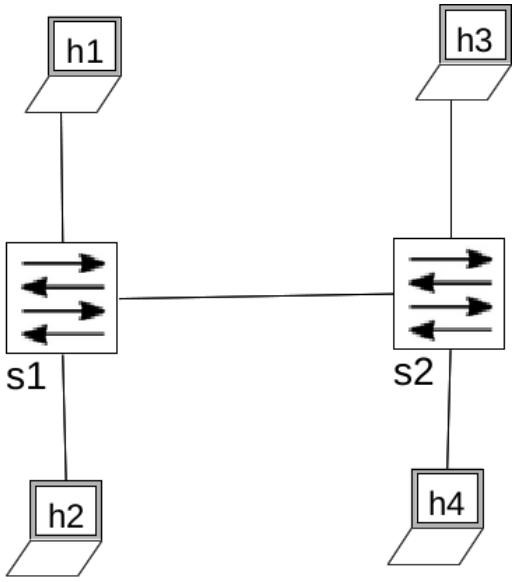
Příloha E

Ďalšie dostupné demonštračné topológie

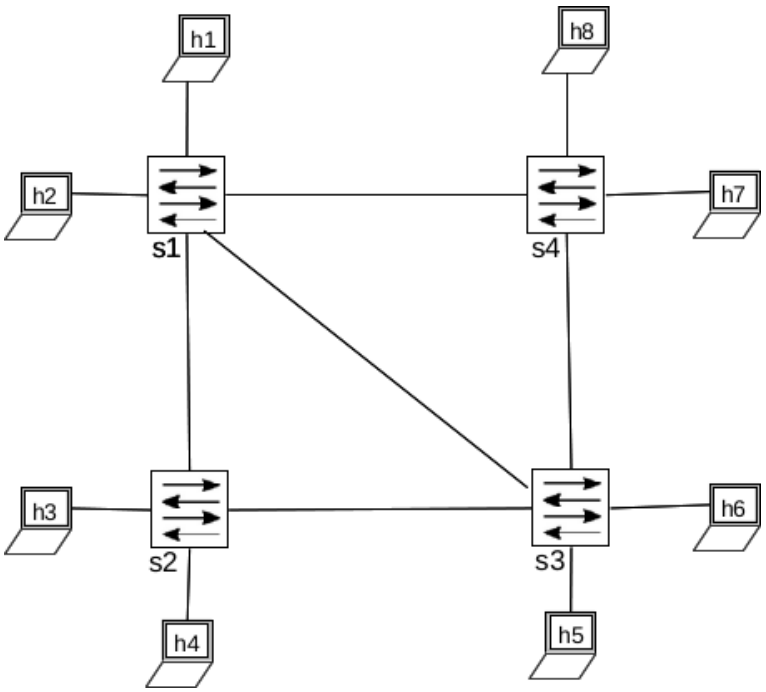
Na CD médiu sú k dispozícii aj ďalšie topológie pre demonštračné účely. Skript pre ich vygenerovanie pomocou nástroja MiniNet je k dispozícii na CD médiu v súbore `demo/topo.py`. Zdrojové kódy v jazyku OpenFlow-Network Control Language sa nachádzajú v adresári `demo/` a sú pomenované podľa príslušnej sieťovej topológie. V hlavičke týchto zdrojových súborov je uvedená aj požadovaná konfigurácia siete. Názvy demonštračných topológií sa nachádzajú v popise obrázkov E.1, E.2 a E.3.



Obrázek E.1: Topológia *top1*. Obsahuje štyri pracovné stanice pripojené k jednému OpenFlow switchu.



Obrázek E.2: Topológia *top2*. Obsahuje štyri pracovné stanice, pripojené k dvom, navzájom prepojeným OpenFlow switchom.



Obrázek E.3: Topológia *top3*. Obsahuje osem pracovných staních, pričom vždy dvojica je pripojená k jednému zo štyroch OpenFlow switchov. OpenFlow switche sú prepojené podľa obrázku.

Příloha F

Obsah CD

Súčasť tejto práce tvorí CD médium, na ktorom sú umiestnené zdrojové kódy implementovaného interpreta jazyka OpenFlow Network Control Language. Obsahuje aj príklady a zdrojové kódy na otestovanie implementácie.

`./doc/`
Táto zložka obsahuje zdrojové kódy dokumentácie pre program LaTeX. Preložená textová časť bakalárskej práce je k dispozícii v súbore `doc/projekt.pdf`.

`./netctrl/`
Zložka obsahuje zdrojové kódy implementovaného interpreta jazyka OF-NCL a OpenFlow kontroléra, súčasťou je aj skript `netctrl`.

`./demo/`
Ukážkové zápisy zdrojových kódov v jazyku OF-NCL pre rôzne typy topológií. Skripty pre vygenerovanie topológií pre program MiniNet sú k dispozícii v súbore `sources/topo.py`.

`./README.txt`
Textový súbor s popisom obsahu CD média.