UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Jan Mokráček

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Analýza kontrolérů softwarově definovaných sítí

Jan Mokráček

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan Mokráček**
Osobní číslo:	**I14146**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Analýza kontrolérů softwarově definovaných sítí**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce bude provést analýzu kontrolérů pro softwarově definované sítě. Autor v teoretické části popíše architektury vybraných kontrolérů a porovná jejich základní vlastnosti a vhodné oblasti nasazení. V praktické části autor popíše implementaci těchto kontrolérů v emulačním nástroji Mininet. Autor vytvoří jednoduchou SDN aplikaci, na které porovná náročnost jejího vývoje v jednotlivých kontrolérech.

Rozsah grafických prací:

Rozsah pracovní zprávy: **35**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

Siamak Azodolmolky, Software Defined Networking with OpenFlow, ISBN-13: 978-1849698726

Thomas D. Nadeau, Ken Gray, SDN: Software Defined Networks, ISBN-13: 978-1449342302

Paul Goransson, Chuck Black, Software Defined Networks: A Comprehensive Approach, ISBN-13: 978-0124166752

Vedoucí bakalářské práce: **Ing. Filip Holík**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2016**
Termín odevzdání bakalářské práce: **12. května 2017**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 7. 5. 2017

Jan Mokráček

PODĚKOVÁNÍ

Mé poděkování patří Ing. Filipovi Holíkovi za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval.

ANOTACE

Práce se zabývá analýzou open source kontrolérů softwarově definovaných sítí. Stručně jsou popsány principy SDN. U jednotlivých kontrolérů je popsána jejich architektura, poskytované vlastnosti, požadavky na hardware a software, instalace a spouštění. V praktické části je popsán vývoj aplikace komunikující se všemi analyzovanými kontroléry včetně rozhraní REST použitého ke komunikaci, formátu JSON použitého k přenosu dat a síťového emulátoru Mininet pro testování aplikace na virtualizované topologii.

KLÍČOVÁ SLOVA

SDN, softwarově definované sítě, kontrolér, REST, JSON, aplikace

TITLE

Analysis of SDN controllers

ANNOTATION

The thesis deals with the analysis of open source software defined networks controllers. It briefly describes the principles of SDN. The architecture, provided features, hardware and software requirements, installation and startup are described for each controller individually. The practical part describes the development of the application which communicates with all the analyzed controllers, including the description of the REST interface which is used for communication, the JSON format which is used for data transfer, and the network emulator Mininet for testing of the application on a virtualized topology.

KEYWORDS

software defined networks, SDN, controller, REST, JSON, application

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

AAA	Authentication, Authorization and Accounting
ACL	Access Control List
ALTO	Application-Layer Traffic Optimization
ARP	Address Resolution Protocol
BGP-LS	Border Gateway Protocol – Link State
BGP-MP	Multi Border Gateway Protocol
BPDU	Bridge Protocol Data Unit
CAPWAP	Control And Provisioning of Wireless Access Points
CoAP	Constrained Application Protocol
CRUD	Create, Read, Update, Delete
CLI	command line interface
cURL	command line tool and library
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DiffServ	Differentiated services
DLUX	OpenDaylight User Experience
DNS	Domain Name System
DSCP	Differentiated Services Code Point
DTLS	Datagram Transport Layer Security
EID	Identifikátor koncového bodu
FaaS	Fabric as a Service
GBP	Group Based Policy
GMPLS	Generalized MPLS

HTTP	Hyper Text Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IoTDM	Internet of Things Data Management
IP	Internet Protocol
ISIS	Intermediate System to Intermediate System
JCL	Java Class Library
JDK	Java Development Kit
JVM	Java Virtual Machine
JSON	Java Script Object Notation
L2	linková vrstva ISO/OSI modelu
L3	síťová vrstva ISO/OSI modelu
LACP	Link Aggregation Control Protocol
LINQ	Language Integrated Query
LISP	The Locator/ID Separation Protocol
LLDP	Link Layer Discovery Protocol
LSP	Label Switched Paths
LTS	Long Term Support
MAC	Media Access Control
MOM	Message-Oriented Middleware
MPLS	Multiprotocol Label Switching
MQTT	MQ Telemetry Transport
NAT	Network Address Translation

NeMo	Network Modeling
NETCONFT	Network Configuration Protocol
oar	ONOS Application aRchive
ODL	OpenDaylight
ONF	Open Networking Foundation
ONOS	Open Networking Operating System
OSPF	Open Shortest Path First
OVS	Open vSwitch
OVSDB	Open vSwitch Database
PCC	Path Computation Client
PCE	Path Computation Element
PCEP	Path Computation Element Protocol
PHB	Per-Hop Behaviour
POM	Project Object Model
REST	Representational State Transfer
RLOC	Routing Locator
SDN	Softwarově definované sítě
SFC	Service Function Chaining
SGT	Source Group Tag
SNBI	Secure Network Bootstrapping Interface
SNMP	Simple Management Network Protocol
SQL	Structured Query Language
STP	Spanning Tree Protocol
TCP	Transmission Control Protocol

TLS	Transport Layer Security
ToS	Type of Service
TSDR	Time Series Data Repository
UDP	User Datagram Protocol
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USC	Unified Secure Channel
UsecPlugin	Unified-Security Plugin
VPN	Virtual Private Network
VLAN	Virtual Local Area Network
XML	eXtensible Markup Language

ÚVOD

Počítačové sítě se, na rozdíl od ostatních oblastí informačních technologií, od doby svého vzniku příliš nezměnily. Současná síťová architektura stále vychází z původních požadavků na ARPANET, který byl navržen pro armádu v roce 1969. Skládá se z decentralizovaných síťových prvků, z nichž každý obsahuje vlastní inteligenci a je plně autonomní. Vzhledem k obrovskému rozvoji počítačových sítí naráží tento přístup již na své limity. Týká se to hlavně sítí s obrovským množstvím zařízení – datových center a poskytovatelů. Možné řešení stávajících nedostatků představují softwarově definované sítě.

Softwarově definované sítě představují zcela nový evoluční přístup k počítačovým sítím. Jejich architektura umožňuje přesunutí inteligence jednotlivých síťových prvků do jednoho centralizovaného prvku, který se v terminologii SDN (softwarově definované sítě) nazývá kontrolér.

Cílem teoretické části této práce je analýza vybraných open source kontrolérů. První kapitola bude věnována stručnému popisu principů a architektury SDN a jejich prvního standardu, protokolu OpenFlow sloužícího ke komunikaci kontroléru s jednotlivými zařízeními. Následující čtyři kapitoly budou věnovány jednotlivým analyzovaným kontrolérům. U každého kontroléru bude popsána jeho architektura, dodávané funkcionality, instalace, obsluha, vyvstanuvší problémy a jejich možná řešení.

V praktické části bude popsána realizace aplikace umožňující zobrazení souhrnných informací o síťové topologii spravované libovolným z analyzovaných kontrolérů. Před popisem samotné aplikace bude také popsáno rozhraní REST (Representational State Transfer) používané aplikací ke komunikaci s kontroléry, formát JSON (Java Script Object Notation) používaný pro přenos dat a síťový emulátor Mininet použitý k testování aplikace.

1 ÚVOD DO SOFTWAROVĚ DEFINOVANÝCH SÍTÍ

Softwarově definované sítě umožňují programovat jednotlivá síťová zařízení a řídit chování sítě jako celku. Termín softwarově definovaná síť byl poprvé použit teprve v roce 2009, rok po vzniku protokolu OpenFlow. Výrazný rozvoj a uplatnění softwarově definovaných sítí na trhu nastává v roce 2011 se založením Open Networking Foundation (ONF), neziskové organizace za jejímž vznikem stojí Deutsche Telekom, Facebook, Google, Microsoft, Yahoo! a Verizon (Goransson & Chuck, c2014, s. 51; Internet Research Task Force, 2015a; Open Networking Foundation, c2017).

1.1 Základní charakteristiky SDN

Klasické počítačové sítě se skládají ze zařízení, z nichž každé obsahuje následující tři vrstvy:

- **Řídící vrstvu** obsahující logiku, algoritmy a protokoly sloužící k programování datové vrstvy. Představuje inteligenci zařízení.
- **Datovou vrstvu** skládající se z portů (které se používají k přijetí a přenosu zpráv) a ze směrovací tabulky. Slouží k vlastnímu přenosu dat.
- **Vrstvu pro správu**, sloužící administrátorovi ke konfiguraci a monitorování jednotlivých zařízení.

Každá z vrstev je schopna komunikovat horizontálně se stejnou vrstvou na sousedním zařízení a vertikálně se sousední vrstvou. Každé zařízení klasické počítačové sítě tedy obsahuje vlastní inteligenci a je plně autonomní. Jednotlivá zařízení spolu komunikují pomocí různých směrovacích protokolů (např. OSPF) a vytváří tak logickou topologii sítě (Goransson & Chuck, c2014, s. 7; Internet Research Task Force, 2015a).

U softwarově definovaných sítí je řídící vrstva přesunuta z hardwaru jednotlivých zařízení do jednoho centralizovaného kontroléru. Veškerá zařízení jsou propojena s tímto kontrolérem a nejsou tedy autonomní. Tím pádem není zapotřebí mít v síti o n-uzlech n-inteligentních zařízení. Řídící vrstva je implementována v softwaru a umožňuje tak programovatelnost sítě a její flexibilnější správu. Administrátorovi je tak umožněna správa sítě jako celku bez nutnosti individuální konfigurace každého zařízení zvlášť pomocí jeho příkazové řádky. Cílem rozdělení vrstev a související programovatelnosti je snížení složitosti zařízení a umožnění rychlejší inovace obou vrstev (Goransson & Chuck, c2014, s. 59–60; Rouse, 2013).

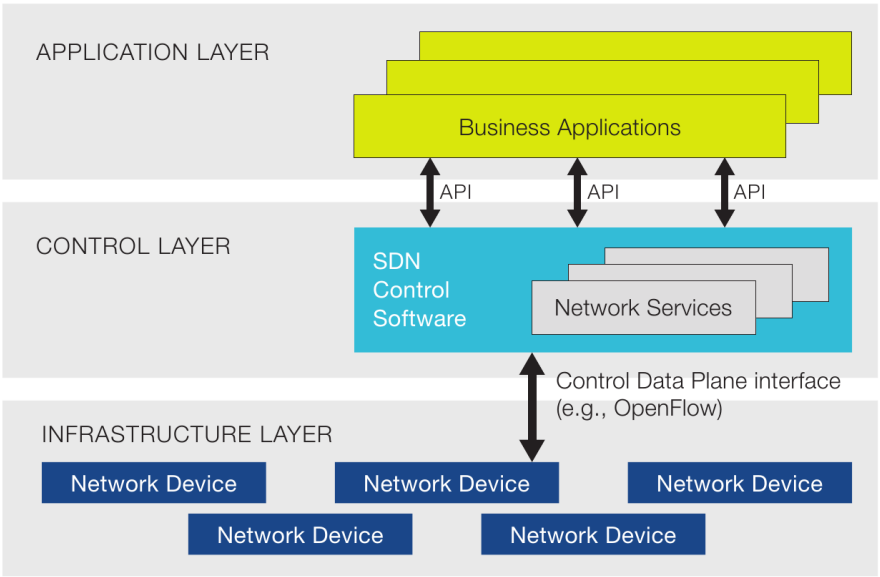
Architektura softwarově definovaných sítí poskytuje následující druhy abstrakcí:

- **Abstrakci distribuovaného stavu** poskytující programátorovi globální pohled na síť. Programátor tím pádem nemusí řešit, že síť se ve skutečnosti skládá z mnoha kooperujících zařízení.
- **Směrovací abstrakci** umožňující specifikovat směrování jednotlivých paketů bez znalosti konkrétního hardwaru, který dané operace provádí.
- **Konfigurační abstrakci** umožňující vyjádření cílů sítě bez nutnosti zabývat se detaily jejich implementace. Je možné použít analogii se zápisem na disk. Programátoři pracují se soubory nikoliv přímo s pevným diskem.

Centralizovaný kontrolér poskytuje otevřená rozhraní sloužící k automatizované správě sítě. Tato rozhraní by měla být dobře dokumentovaná, standardizovaná a neměla by být proprietární. Otevřenost softwarově definovaných sítí umožňuje jejich rychlejší rozvoj a interoperabilitu zařízení od různých výrobců (Goransson & Chuck, c2014, s. 60–61).

1.2 Architektura SDN

Architektura SDN se skládá ze tří vrstev, vizte obrázek 1. Vrstvy spolu vzájemně komunikují pomocí rozhraní, která budou popsána v kapitole SDN kontrolér.



Obrázek 1 – Architektura softwarově definovaných sítí

Zdroj: (SDNCentral, c2012–2017a)

Aplikační vrstva se skládá z aplikací běžících nad kontrolérem. Jednotlivé aplikace programově řídí síť pomocí kontroléru. Chování jednotlivých aplikací je řízeno událostmi

přijatými od kontroléru. Aplikace naslouchá určitým událostem (např. kontrolér přijal paket), na které reaguje vyvoláním určitých metod a tímto způsobem ovlivňuje celou síť (Goransson & Chuck, c2014, s. 72–73; SDNCentral, c2012–2017a; Open Networking Foundation, c2013).

Řídící vrstva funguje jako prostředník mezi aplikační a infrastrukturní vrstvou. Předává instrukce a požadavky od aplikační vrstvy jednotlivým síťovým prvkům a upozorňuje aplikace na vzniklé síťové události (SDNCentral, c2012–2017a).

Infrastrukturní vrstva se skládá z jednotlivých SDN zařízení, jejichž cílem je zpracování příchozích paketů. Paket může být zpracován lokálně, pokud je nalezena shoda, anebo může být přeposlán řídící vrstvě a poté zpracován některou z aplikací (Goransson & Chuck, c2014, s. 64–68; SDNCentral, c2012–2017a).

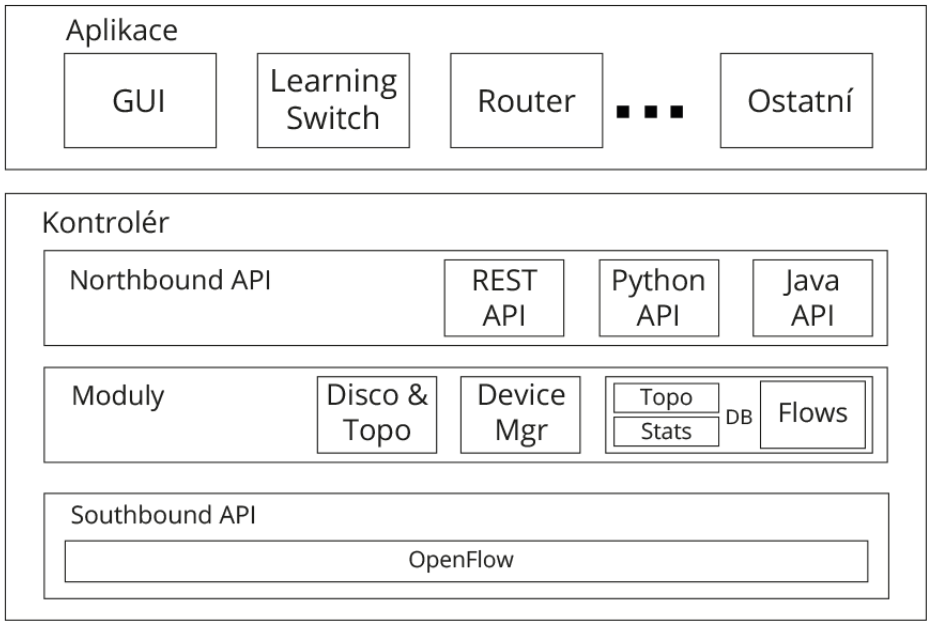
1.3 SDN kontrolér

Kontrolér je software, který udržuje pohled na celou topologii sítě, implementuje síťovou politiku a řídí všechna SDN zařízení v topologii. Poskytuje *northbound* a *southbound* API, jejichž názvy jsou odvozeny od světových stran a odkazují na jejich pozici vzhledem k jádru kontroléru (Goransson & Chuck, c2014, s. 68–69). Architektura SDN kontroléru je znázorněna na obrázku 2.

Southbound API, někdy též označováno jako *Control to Data-Plane Interface*, slouží k programování síťových zařízení. Toto rozhraní je v současné době velmi dobře standardizováno (Goransson & Chuck, c2014, s. 70; SDNCentral, c2012–2017a). Příkladem takového rozhraní je standard OpenFlow nebo OVSDB (Open vSwitch Database).

Northbound API umožňuje začlenit aplikace do kontroléru. Poskytuje aplikacím pohled na celou topologii a abstrahuje tak od detailů jednotlivých zařízení. Algoritmy a protokoly implementované v aplikacích mohou díky této abstrakci efektivně řídit síť (Goransson & Chuck, c2014, s. 70).

Veškeré funkcionality kontroléru (např. *routing*, *firewall*) jsou implementovány jako aplikace nebo moduly. Základní funkcionality, jako je například *learning switch*, jsou ve většině případů dodávány společně s kontrolérem ve formě modulů. Na obrázku 2 jsou vyobrazeny moduly poskytující nejzákladnější funkce, *southbound a northbound* API a několik příkladů aplikací, které využívají kontrolér (Goransson & Chuck, c2014, s. 72–73).



Obrázek 2 – Architektura SDN kontroléru

Zdroj: zpracováno podle (Goransson & Chuck, c2014, s. 69)

1.3.1 Základní moduly kontroléru

Kontrolér abstrahuje detaily *southbound* protokolů (např. OpenFlow) a díky tomu umožňuje aplikacím komunikovat s jednotlivými zařízeními v síti, aniž by znaly jejich nuance. Každý kontrolér obsahuje sadu interních modulů, které implementují základní funkcionality mezi *southbound* a *northbound* API. Mezi tyto základní funkcionality patří:

- zjišťování koncových zařízení (např. notebooky, mobilní zařízení atd.),
- zjišťování síťových zařízení (např. přepínače, routery, bezdrátové přístupové body),
- udržování informací ohledně spojení mezi jednotlivými síťovými a koncovými zařízeními,
- udržování informací o tocích v databázi spravovaných kontrolérem a zajištění synchronizace flow záznamů na zařízeních s databází.

Jak ukazuje obrázek 2, tyto interní moduly si udržují lokální databázi, která obsahuje informace o topologii a statistiky. Kontrolér se učí topologii sítě zjišťováním aktivních prvků a koncových zařízení a sledováním konektivity mezi nimi (Goransson & Chuck, c2014, s. 70).

1.4 OpenFlow

Podkapitola, pokud není uvedeno jinak, čerpá z knihy Paula Göranssona a Chucka Blacka (Goransson & Chuck, c2014, s. 81–116).

OpenFlow je první neproprietární protokol sloužící k programování datové vrstvy. Jedná se o standardní protokol definující způsob komunikace mezi datovou a řídící vrstvou. OpenFlow protokol definuje zprávy a jejich formát, které si mezi sebou vyměňují kontrolér a SDN zařízení. Specifikuje, jak budou SDN zařízení reagovat na určité situace a jak by měla odpovídat na příkazy od kontroléru.

Klasická síťová zařízení mají informace uložené v různých tabulkách (např. MAC tabulka, směrovací tabulka atd.). Pomocí těchto tabulek je následně naprogramována datová vrstva. U OpenFlow zařízení jsou veškeré tabulky nahrazeny flow tabulkou.

Flow tabulka se skládá z políček hlavičky (kritérium shody pro příchozí pakety), čítačů a akcí. Těchto tabulek se na zařízení může nacházet i více. Jednotlivé záznamy jsou seřazeny dle priority a mohou odkazovat na další tabulky. Tímto způsobem je zvýšena flexibilita a modifikovatelnost políček hlavičky paketu. Při každé shodě dochází k nějaké akci.

OpenFlow zařízení mohou komunikovat s kontrolérem zabezpečeným kanálem pomocí OpenFlow zpráv (např. *hello, packet-in, packet-out*). Ve výchozím nastavení však není komunikace zašifrována. Pomocí *packet-in* přeposílá zařízení pakety kontroléru, kde jsou poté zpracovány určitými aplikacemi nebo moduly. Pomocí *packet-out* posílá kontrolér zpět přijaté (*packet-in*) pakety zařízením. Pomocí *flow-add, flow-remove, flow-mod* kontrolér vkládá, maže a upravuje flow záznamy v tabulkách jednotlivých zařízení.

Zařízení hledá pro přijatý paket záznam ve flow tabulce. Při shodě zařízení provede jednu z následujících akcí:

- Pošle paket skrze lokální port dále a případně provede modifikaci určitých políček hlavičky.
- Zahodí paket.
- Pošle paket ke zpracování kontroléru (*packet-in*).

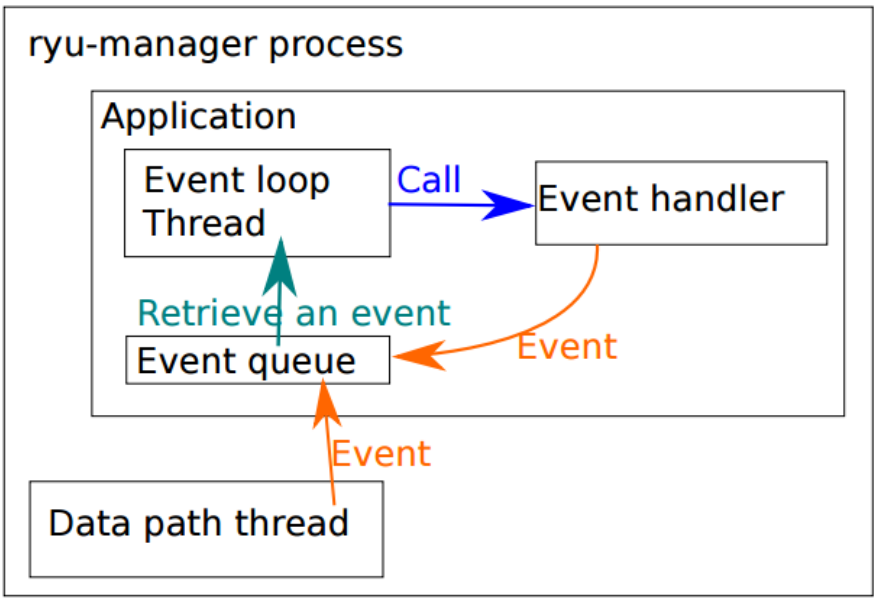
2 RYU

Kapitola, pokud není uvedeno jinak, čerpá z oficiální dokumentace Ryu kontroléru (RYU project team, c2014a).

Ryu (v japonštině znamená tok) je framework pro SDN založený na komponentách (moduly a aplikace). Poskytuje softwarové komponenty s dobře definovanými API, které umožňují vývojářům snadno vytvořit nové způsoby správy sítě a nové řídící aplikace. Podporuje několik různých protokolů pro správu síťových zařízení jako OpenFlow, NETCONF a další. Veškerý kód je volně dostupný pod Apache 2.0 licencí. Ryu kontrolér je celý naprogramovaný v jazyce Python a podporuje tedy moduly napsané pouze v tomto jazyce (Nippon Telegraph and Telephone Corporation, c2011–2014a).

2.1 Architektura

V této kapitole bude představena architektura Ryu kontroléru. Přesněji řečeno programovacího modelu, který využívají aplikace naprogramované pro Ryu kontrolér. Veškeré aplikace potažmo moduly Ryu kontroléru je nutné spouštět skrze tzv. manažera, který se nachází ve složce `bin/` pod názvem *ryu-manager*. Tento manažér dále poskytuje aplikacím kontext a umožňuje směrování zpráv mezi jednotlivými aplikacemi. Jedná se o skript napsaný v jazyce Python. Na obrázku 3 jsou znázorněny součásti programovacího modelu, které využívají aplikace Ryu kontroléru.



Obrázek 3 – Programovací model

Zdroj: (RYU project team, c2014b)

Programovací model, který využívají aplikace Ryu kontroléru se skládá z následujících součástí:

- Aplikace je třída dědící z `ryu.base.app_manager.RyuApp`.
- Událost je objekt třídy, která dědí z `ryu.controller.event.EventBase`. Umožňuje komunikaci mezi jednotlivými aplikacemi. Komunikace funguje na principu posílání a přijímání událostí.
- Fronta událostí sloužící k dočasnému ukládání událostí přijatých aplikací. Události jsou z fronty vybírány v pořadí, v jakém do ní byly uloženy.
- Ryu běží ve více vláknech a používá tzv. *eventlets*. *Eventlets* je síťová knihovna umožňující měnit způsob běhu kódu. Jednotlivá vlákna nejsou preemptivní. Nevzdají se tedy procesoru až do chvíle, kdy jsou ukončena. Z tohoto důvodu je nutné dávat pozor na procesy, které se provádí delší časový úsek. Pro každou aplikaci je automaticky vytvořeno jedno vlákno, na kterém běží smyčka událostí. Smyčka událostí slouží k výběru přijatých událostí z fronty a zavolání příslušných obslužné rutin těchto událostí. Pro každou aplikace je také možné vytvořit další přídavná vlákna, která budou sloužit k vykonání specifických zpracování.
- Obslužné rutiny událostí. Pomocí `ryu.controller.handler.set_ev_cls` je možné definovat vlastní obslužné rutiny událostí tak, že se metoda třídy aplikace upraví tzv. dekorátorem. Poté, když nastane specifická událost, tak je zavolána příslušná obslužná rutina pomocí smyčky událostí (RYU project team, c2014b).

2.2 Instalace

Ryu jako jediný z analyzovaných kontrolérů není založený na Javě, nýbrž na jazyce Python. Pro jeho instalaci potažmo spuštění, stačí mít nainstalovaný pouze *pip*. Pip je systém pro instalaci a správu balíčků napsaných v Pythonu. Oproti kontrolérům ONOS a OpenDaylight má Ryu velice nízké nároky na operační paměť. S pouhými 256 MiB rezervovanými pro virtuální stroj získává kompletní pohled na topologii již po 30 vteřinách. Ryu kontrolér je možné instalovat pouze na operačním systému Linux. Preferována je nejnovější LTS (Long Term Support) distribuce. Níže je uveden návod k instalaci Ryu na distribuci Ubuntu (16.04).

Příkaz pro instalaci programu git:

```
#sudo apt install git
```

Příkaz sloužící ke zkopírování zdrojového kódu Ryu do domovské složky uživatele:

```
#git clone git://github.com/osrg/ryu.git
```

Příkaz sloužící k instalaci systému pro instalaci a správu balíčků napsaných v Pythonu:

```
#pip install
```

Příkaz sloužící k instalaci samotného Ryu, za předpokladu, že se uživatel nachází ve složce, která obsahuje jeho zdrojový kód:

```
#pip install .
```

2.3 Spouštění

Ryu se spouští skrze tzv. manažera[1], který slouží uživateli ke správě aplikací a modulů. Umožňuje mimo jiné specifikaci spouštěných aplikací či modulů, jejich ladění *(debug)* a zobrazení událostí týkajících se zjišťování topologie. Na rozdíl od ostatních analyzovaných kontrolérů je nutné specifikovat využívané aplikace či moduly při každém startu kontroléru pomocí příkazu. Příkaz pro spuštění jednotlivých modulů či aplikací se značně liší od příkazu, který je zapotřebí použít v případě, že chce uživatel využívat více modulů či aplikací zároveň. Libovolný výstup od aplikací a modulů se vždy objevuje v terminálu ve kterém byl kontrolér spuštěn.

Příkaz pro spuštění kontroléru s jedním modulem či aplikací:

```
#ryu-manager [zobrazení událostí/ladění] [cesta k aplikaci]
```

Příkaz pro spuštění kontroléru s více aplikacemi či moduly:

```
#PYTHONPATH=. [cesta k manažeru správy aplikací] [zobrazení událostí/ladění] [cesta k první aplikaci] [cesta k druhé aplikaci] …
```

[1] ryu-manager --help vypíše veškeré přepínače

2.4 Moduly

V tabulce 1 jsou uvedeny moduly dodávané společně s Ryu kontrolérem. Některé z uvedených modulů poskytují REST API sloužící uživateli k ovlivňování sítě nebo k vyžádání informací (např. flow záznamy na zařízení). Uživatel ke komunikaci s REST API může využít například cURL (command line tool and library). Veškerá poskytována REST API je možné nalézt v oficiální dokumentaci Ryu kontroléru.[2]

Tabulka 1 – Moduly dodávané společně s Ryu kontrolérem

Modul	Název modulu ve složce Ryu kontroléru*
Agregace spojení	simple_switch_lacp_13.py
Firewall#	rest_firewall.py
Kvalita služeb#	rest_qos.py
L2 přepínač	simple_switch.py, simple_switch_12.py, simple_switch_13.py, simple_switch_14.py, example_switch_13.py
L2 přepínač#	simple_switch_rest_13.py
Monitoring síťového provozu	simple_monitor_13.py
OFCTL#	ofctl_rest.py
Router#	rest_router.py
Spanning tree	simple_switch_stp.py, simple_switch_stp_13.py
Testovací nástroj OpenFlow přepínačů	tester.py

*číslo za názvem modulu označuje pro jakou verzi OpenFlow je určen

#modul poskytující REST API

Agregace spojení umožňuje sdružení n-fyzických spojení do jednoho logického. Například protokol STP poté vidí těchto několik fyzických spojení jako jedno. Agregace spojení umožňuje vyšší komunikační rychlost mezi jednotlivými zařízení a vyvážení zatížení mezi jednotlivými fyzickými spoji. Zajišťuje také redundanci spojení a poskytuje tak větší odolnost vůči chybám. Když selže jedno z fyzických spojení, tak to neznamená selhání celého logického spojení. Modul k agregaci spojení využívá protokolu LACP (Link Aggregation Control Protocol). Tento modul je opět nadstavbou L2 přepínače, který zajišťuje zpracovávání paketů.

Firewall poskytuje REST API k nastavení pravidel pro síťovou komunikaci. Firewall zde funguje jako klasický ACL (Access Control List). Jednotlivá pravidla jsou umístěna v seznamu a mají nastaveny priority určující, které pravidlo se provede jako první. Pravidla je

[2] https://osrg.github.io/ryu-book/en/Ryubook.pdf

možné pouze vkládat a mazat, nikoli upravovat. Jako kritérium shody je možné použít IPv4 (Internet Protocol) adresy a VLAN. Pokud je přijat paket, který je zablokován, tak se tato událost objeví v terminálu, ve kterém byl spuštěn kontrolér.

Kvalita služeb umožňuje přenášení dat v souladu s prioritou založenou na typu dat a rezervaci síťového pásma pro určitou komunikaci. Umožňuje poskytování služeb s předem garantovanou kvalitou. Díky tomu nedochází ke ztrátám, zpožděním nebo plýtvání s dostoupnou šířkou pásma. Pro nastavení kvality služeb slouží poskytovaná REST API. Modul nabízí následující typy kvality služeb:

- Kvalita služeb na bázi přidávání flow záznamů do určitých vytvořených front. Umožňuje jemnější řízení kvality služeb. Nicméně s rozšiřováním sítě se zvyšuje i počet flow záznamů, které zařízení potřebuje k řízení kvality služeb. Každý takovýto záznam je potřeba instalovat do zařízení manuálně.
- Kvalita služeb na bázi používání DiffServ (Differentiated services). Rozděluje flow záznamy do několika tříd na vstupním routeru DiffServ domény a aplikuje DiffServ k řízení flow záznamů jednotlivých tříd. DiffServ odesílá pakety podle PHB (Per-Hop Behaviour), definovaného hodnotou DSCP (Differentiated Services Code Point). Jedná se o prvních šest bitů pole ToS (Type of Service), které se nachází v hlavičce IP paketu.
- Kvalita služeb na bázi používání meter tabulky. Meter tabulka byla představena v OpenFlow verzi 1.3 a umožňuje nastavení kvality služeb pomocí OpenFlow.

L2 přepínač poskytuje funkcionalitu L2 (linková vrstva ISO/OSI modelu) směrování skrze připojené OpenFlow přepínače a udržuje přehled o veškerých koncových zařízení sítě. Učí se MAC (Media Access Control) adresy hostů připojených k portům a uchovává je v tabulce MAC adres. Pokud je přijatý paket adresován hostu, jehož MAC adresa je již v tabulce, tak tento paket pošle pouze na port, ke kterému je připojen tento host. Pokud přijme paket, který je adresován neznámému hostu, tak tento paket rozešle na všechny porty. Nicméně tento modul neimplementuje STP (Spanning Tree Protocol) ani jiný algoritmus, který by z topologie odstraňoval smyčky. Může docházet k broadcastovým bouřím.

L2 přepínač s REST API má oproti modulu L2 přepínač, popsanému výše, přidána dvě následující REST API:

- Rozhraní pro získání tabulky MAC adres konkrétního přepínače. Obsah MAC tabulky je vrácen ve formátu JSON.
- Rozhraní pro registraci MAC adresy do tabulky MAC adres sloužící k přidání flow záznamu do flow tabulky daného přepínače. Kritérium shody je příslušná MAC adresa a akcí je poslání paketu skrze určený lokální port.

Monitoring síťového provozu slouží, jak již napovídá název, k monitorování provozu SDN sítě. Sbírá v konstantním intervalu informace o stavu sítě. Například informace o změnách stavu jednotlivých portů atd. Získává informace týkající se jednotlivých flow záznamů, kromě tzv. *table miss*. *Table miss* je poslední záznam ve flow tabulce s nejnižší prioritou, shodný s libovolným paketem, který přepínač přijme. Tento modul je nadstavbou L2 přepínače a obsahuje veškeré jeho funkcionality. Není tedy zapotřebí spouštět společně s tímto modulem ještě L2 přepínač. K zajištění souběžného zpracovávání paketů a periodického dotazování OpenFlow přepínačů se používá více vláken. Každá z těchto funkcionalit běží na samostatném vlákně.

Modul **OFCTL** poskytuje uživateli REST API, která mu umožňují:

- Získávání stavů jednotlivých přepínačů, souhrnných informací o topologii a různých statistik.
- Modifikaci různých aspektů konkrétního zařízení (např. přidání flow záznamu, změna role).

Seznam poskytovaných REST API modulem *ofctl* je možné nalézt přímo na oficiálních stránkách kontroléru Ryu[3] (Nippon Telegraph and Telephone Corporation, c2011–2014b).

Router poskytuje REST API sloužící k nastavení L3 směrování. Umožňuje uživateli nastavení IP adres a VLAN na jednotlivých rozhraních zařízení, nastavení statického směrování a nastavení výchozí cesty.

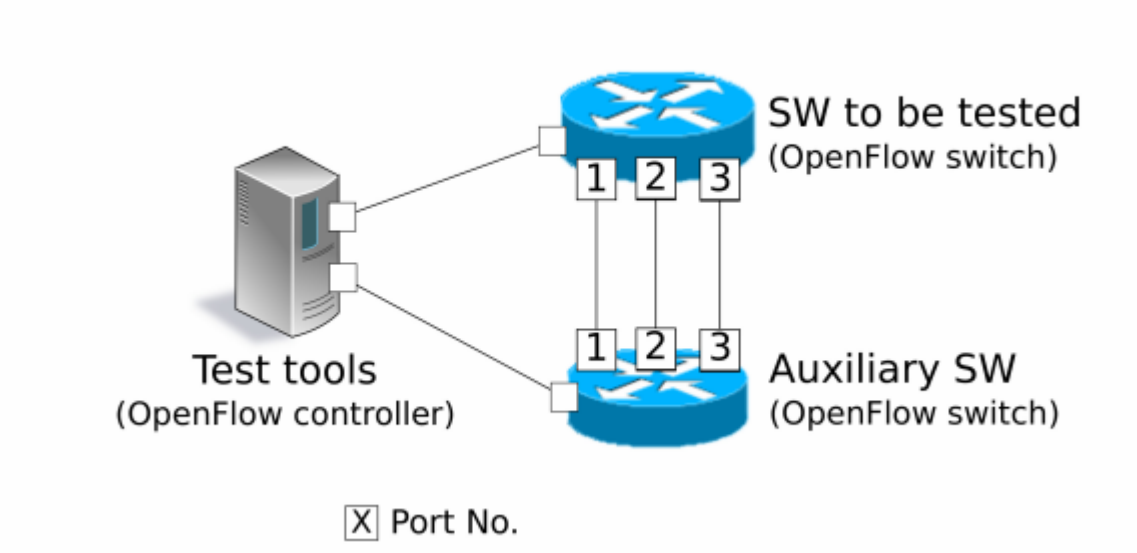
Spanning tree je nadstavbou modulu L2 přepínače. Navíc obsahuje implementaci STP protokolu. STP protokol vytváří ze sítě logický strom a odstraňuje tak smyčky. Logický strom vytváří tak, že na jednotlivých přepínačích přepíná stavy portů (*forward/block*). Jednotlivé přepínače si mezi sebou vyměňují informace jak o sobě, tak o svých portech pomocí

[3] http://ryu.readthedocs.io/en/latest/app/ofctl_rest.html

BPDU (Bridge Protocol Data Unit) paketů a na základě těchto informací se poté rozhoduje, zdali danému portu bude umožněno přenášení rámců. STP protokol tak zabraňuje *broadcastovým bouřím* a zajišťuje stabilitu tabulky MAC adres. Modul implementuje pouze základní STP, neboť STP knihovna v současné době nepodporuje VLAN (Virtual Local Area Network).

Testovací nástroj OpenFlow přepínačů slouží k verifikaci podpory verze OpenFlow určitým přepínačem. Verifikace se provádí pomocí flow a meter záznamů do testovaného OpenFlow přepínače podle vzorového testovací souboru a porovnání výsledného zpracování, přenosu a modifikace paketů vůči očekávaným výsledkům. Očekáváné výsledky jsou popsány ve vzorovém testovacím souboru. Testovací prostředí je zobrazeno na obrázku 4 a skládá se z OpenFlow kontroléru, testovaného přepínače a pomocného přepínače. Pomocný OpenFlow přepínač musí úspěšně provádět následující operace:

- Registraci flow záznamů s definovanou akcí CONTROLLER.
- Registraci flow záznamů s měřením propustnosti.
- Přenos *packet-in* zpráv, pokud paket spadá pod flow záznam s definovanou akci CONTROLLER.
- Přenos paketu přijatého od kontroléru pomocí *packet-out* zprávy.

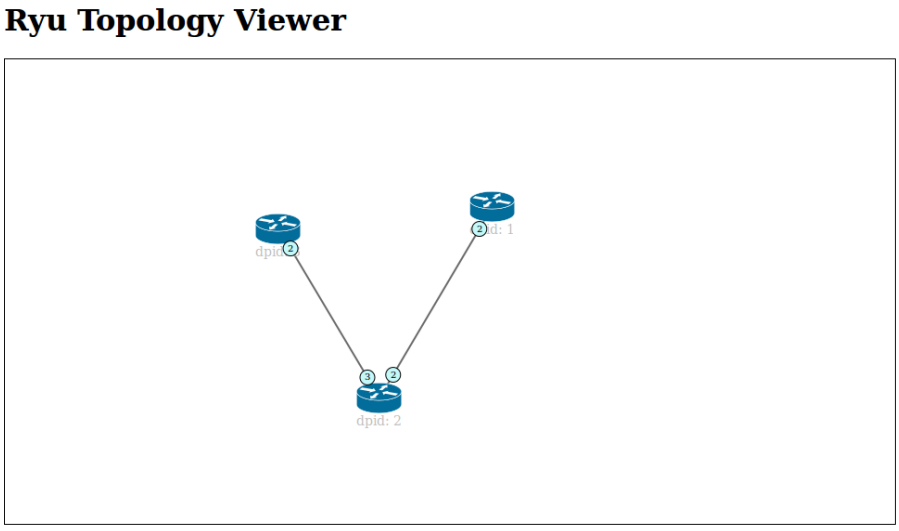


Obrázek 4 – Testovací prostředí

Zdroj: (RYU project team, c2014a)

2.5 Vizualizace topologie

Jedná se o aplikaci sloužící ke zobrazení topologie SDN sítě ve webovém prohlížeči pomocí poskytovaných REST API. Aplikace neumožňuje zobrazení kompletní topologie, včetně koncových zařízení. Zobrazuje pouze přepínače, jejich unikátní identifikátory a spojení mezi nimi. Po kliknutí na konkrétní přepínač dokáže také zobrazit jeho flow záznamy. Příklad vizualizované topologie ve webovém prohlížeči je zobrazen na obrázku 5. Aplikace neobsahuje logiku pro komunikaci mezi zařízeními. Společně s touto aplikací je tedy zapotřebí spustit rovněž modul, který tuto logiku poskytuje. Po spuštění daného modulu ztrácí aplikace schopnost zobrazování flow záznamů.



Obrázek 5 – Ukázka vizualizace topologie

3 OPENDAYLIGHT

Kapitola, pokud není uvedeno jinak, čerpá z oficiální dokumentace kontroléru Opendaylight (OpenDaylight Project, c2016).

Projekt ODL (OpenDaylight) je open source platforma pro softwarově definované sítě, která používá otevřené protokoly, s jejichž pomocí poskytuje centralizované a programovatelné řízení a monitorování síťových zařízení. Platforma podporuje OpenFlow protokol a nabízí různá síťová řešení, která jsou součástí platformy a stačí je pouze nainstalovat. Příkladem takového síťového řešení je logika L2 přepínačů.

ODL umožňuje funkci síťových služeb napříč prostředím tvořeným zařízeními od různých výrobců. Architektura mikroslužeb umožňuje uživatelům řízení aplikací, protokolů a zásuvných modulů a poskytuje spojení mezi externími zákazníky a poskytovateli. Vývoj ODL je tvořen velkou globální komunitou, která aktualizuje platformu zhruba každých šest měsíců a nepřetržitě ji adaptuje tak, aby její užití bylo co nejširší.

ODL kontrolér může běžet jako distribuovaný systém napříč více servery. Umožňuje využívat procesory a paměťové zdroje více serverů a zároveň poskytuje odolnosti vůči chybám, neboť při pádu jednoho serveru nedojde k narušení konektivity. Potenciálně také poskytuje podporu ke změně hardwaru či aktualizaci softwaru za běhu, aniž by byla narušena konektivita.

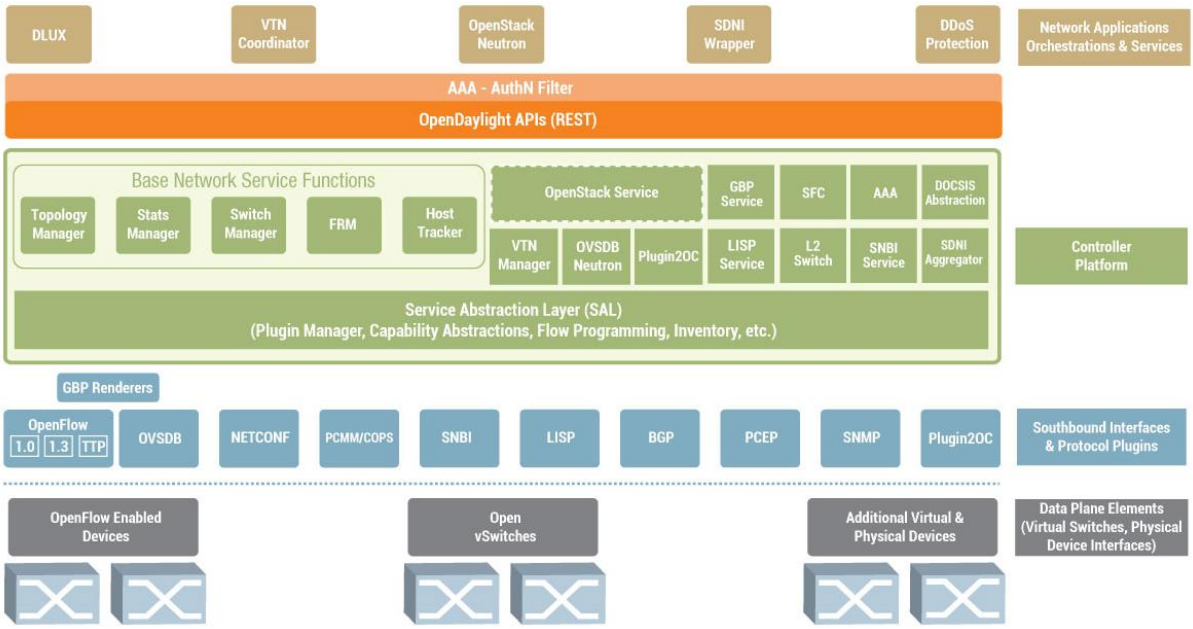
3.1 Architektura

ODL kontrolér je JVM (Java Virtual Machine) software, který může běžet na jakémkoli operačním systému a hardwaru podporujícím Javu. Kontrolér je implementací SDN konceptu a využívá následující nástrojů:

- Maven pro automatizaci buildů aplikací.
- OSGi framework představuje *back-end* ODL, umožňuje dynamické načítání svazků a balíčků JAR souborů a vázání svazků dohromady pro výměnu informací.
- JAVA rozhraní používána pro naslouchání událostí, specifikaci a formování vzorů. To je hlavní způsob, jakým specifické balíčky implementují funkce pro zpětná volání při výskytu událostí a k indikaci povědomí specifického stavu.
- REST API. Přes *northbound* rozhraní poskytuje funkce jako jsou *topology manager*, *host tracker*, *flow programmer*, *static routing* a další.

Kontrolér zpřístupňuje otevřená *northbound* rozhraní, která používají aplikace. OSGi framework a obousměrné REST jsou podporovány *northbound* rozhraními. Architekturu

ODL kontroléru znázorňuje obrázek 6. OSGi framework se používá pro aplikace, které běží ve stejném adresním prostoru jako kontrolér, zatímco REST API je používáno pro aplikace, které běží v jiném adresním prostoru než kontrolér. Doménová logika a algoritmy jsou umístěny v aplikacích. Tyto aplikace využívají kontrolér ke shromažďování inteligence sítě a spouštějí své algoritmy. Pomocí těchto algoritmů provádí analytiku a poté organizují nová pravidla v rámci celé sítě. Na *southbound* rozhraní je podporováno více protokolů ve formě zásuvných modulů, jako je například OpenFlow 1.0, OpenFlow 1.3, BGP-LS (Border Gateway Protocol – Link State) a další. Další ODL přispěvatelé přidávají do kontroléru svůj kód ve formě modulů. Tyto moduly jsou dynamicky propojeny do SAL (Service Abstraction Layer).



Obrázek 6 – Architektura kontroléru OpenDaylight

Zdroj: (OpenDaylight Project, c2016)

SAL zpřístupňuje služby a na jejich základě jsou napsány moduly nad touto vrstvou. SAL zjistí, jak splnit požadované služby bez ohledu na základní protokol používaný mezi kontrolérem a jednotlivými zařízeními. Toto poskytuje ochranu finančních investic do aplikací jako OpenFlow a dalších protokolů, které se v průběhu času vyvíjejí. Aby kontrolér mohl řídit zařízení ve své doméně, musí o zařízeních vědět, znát jejich schopnosti, dostupnost atd. Tyto informace ukládá a spravuje *Topology Manager*. Další komponenty jako je *ARP handler*, *Host Tracker*, *Device Manager* a *Switch Manager* pomáhají *Topology Manageru* ve vygenerování topologické databáze.

3.2 Instalace

ODL kontrolér je, jak již bylo zmíněno výše, založený na jazyce Java a ke svému spuštění tedy vyžaduje JVM (Java Virtual Machine). Nejprve je tady zapotřebí nainstalovat JDK (Java Development Kit), jehož je JVM součástí.

JDK je softwarový balíček, který obsahuje vše potřebné ke spuštění programu napsaného v jazyce Java. Obsahuje implementaci JVM společně s implementací JCL (Java Class Library). JCL je soubor knihoven, které je možné dynamicky načítat aplikacemi za běhu (Javatpoint, c2011–2017).

ODL dokumentace také doporučuje nastavení proměnné `JAVA_HOME`. Jedná se o proměnou prostředí, podle které programy napsané v jazyce Java vyhledávají interpret a kompilér. ODL kontrolér je funkční i bez jejího nastavení.

Při instalaci prostředí Mininet je zcela nezbytné, aby byl ODL kontrolér vypnutý. Jinak bude fungovat pouze při prvním spuštění a při dalším bude vyvolat výjimku, že adresa je již používána. Pro odstranění této chyby bylo nutné kompletně přeinstalovat virtuální stroj.

ODL kontrolér má vysoké nároky na operační paměť. Dokumentace ODL kontroléru uvádí, že minimální požadovaná operační paměť je 2 GiB. Při vyhrazení 2 GiB operační paměti virtuálnímu stroji dochází po spuštění ODL kontroléru k celkovému zpomalení systému. Začne docházet k neustálému swapování a systém se stává takřka nepoužitelným. Důsledkem toho je, že může trvat i několik minut, než se uživateli povede přihlásit k webovému grafickému rozhraní kontroléru. Kontrolér získává kompletní pohled na topologii sítě (3 koncová zařízení a 3 přepínače) přibližně až po 5 minutách. Ideální velikost operační paměti je 4 GiB. Při vyhrazení 4 GiB operační paměti pro virtuální stroj, získává ODL kontrolér kompletní pohled na topologii sítě již za 20 vteřin. ODL kontrolér je možné nainstalovat na libovolný operační systém. Oficiální dokumentace však doporučuje využít nejnovější LTS (Long Term Support) distribuci operačního systému Linux. Níže je uveden návod k instalaci ODL na distribuci Ubuntu (16.04).

Příkaz pro nainstalování JDK[4]:

```
#sudo apt-get install openjdk-8-jdk
```

[4] Instalace JDK a nastavení proměnné `JAVA_HOME` je společné pro všechny kontroléry napsané v Javě, jmenovitě ODL, Floodlight a Open Networking Operating Systém. Z tohoto důvodu nebude tato část návodu u následujících kontrolérů uváděna.

Příkaz pro zjištění cesty, kde je nainstalován balíček JDK:

```
#sudo update-alternatives --config java
```

Editace konfiguračního souboru `/etc/environment` pomocí textového editoru:

```
#sudo nano /etc/environment
```

Přidání následujícího řádku (cesta) do konfiguračního souboru pro nastavení proměnné `JAVA_HOME`:

```
JAVA_HOME=“/user/lib/jvm/java-8-openjdk-amd64“
```

Příkaz sloužící pro aplikaci nastavení proměnné:

```
#source /etc/environment
```

Stažení karaf distribuce Beryllium-SR4 ze stránky *https://www.opendaylight.org/downloads* ve formátu tar.

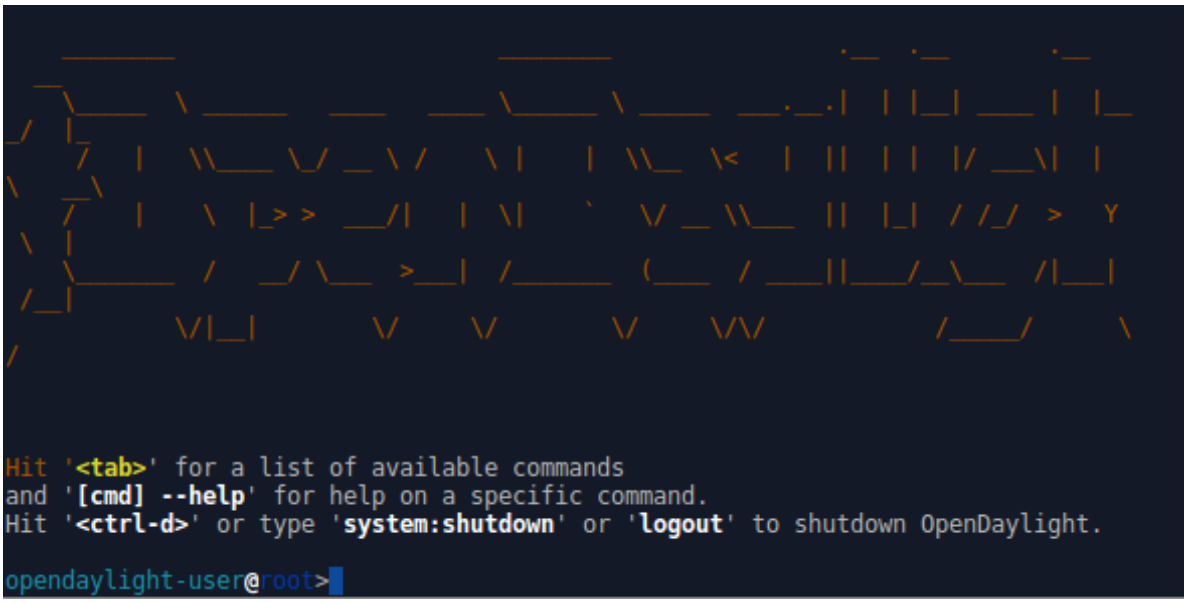
Příkaz sloužící k rozbalení distribuce karaf:

```
tar zxvf distribution-karaf-0.4.4-Beryllium-SR4.tar.gz
```

Po provedení všech výše uvedených kroků byl nainstalován základní ODL kontrolér v takovém stavu, že je možné spustit jeho CLI (command line interface).

3.3 Spouštění

Po samotné instalaci umožňuje ODL kontrolér uživateli jednorázově doinstalovat požadované funkcionality ve formě aplikací skrze aplikační kontejner Karaf (obrázek 7). Doinstalované aplikace jsou poté spouštěny automaticky vždy při startu kontroléru. Je nutné doinstalovat pouze ty aplikace, které uživatel skutečně potřebuje. Čím více aplikací je doinstalováno, tím větší jsou nároky kontroléru na operační paměť. ODL kontrolér nepodporuje aplikace odinstalovat jednotlivě. Pro odstranění jedné problematické aplikace je tedy zapotřebí odinstalovat i všechny ostatní a vrátit tak kontrolér do výchozí nastavení. Tím uživatel ztrácí veškeré nainstalované aplikace a nastavení kontroléru.



Obrázek 7 – Aplikační kontejner Karaf kontroléru OpenDaylight

Příkaz sloužící ke spuštění aplikačního kontejneru Karaf (složka s distribucí ODL):

```
#.bin/karaf
```

Příkaz pro zjištění názvu aplikací dodávaných společně s kontrolérem (CLI kontroléru):

```
#feature:list
```

Příkaz pro jednorázovou instalaci požadované aplikace (CLI kontroléru):

```
#feature:install [nazev_aplikace1] [nazev_aplikace2]
```

Příkaz pro ověření nainstalovaných aplikací (CLI kontroléru):

```
#feature:list -i
```

Příkaz pro vymazání veškerých doinstalovaných aplikací a vrácení kontroléru do výchozího nastavení:

```
#bin/start clean
```

Na rozdíl od kontroléru Floodlight není po změně konfigurace nebo doinstalování aplikace nutné restartovat kontrolér. Kompletní výpis příkazů Karaf CLI je možné nalézt na oficiálních stránkách ODL kontrolérů.[5]

[5] http://docs.opendaylight.org/en/stable-beryllium/getting-started-guide/installing_opendaylight.html

3.4 Vlastnosti

Termín aplikace a modul se používá v dokumentaci ODL kontroléru zaměnitelně. Veškeré moduly a aplikace se souhrnně nazývají vlastnosti. V této podkapitole budou představeny některé z vlastností (tabulka 2) dodávaných společně s ODL kontrolérem. Jednotlivé vlastnosti je možné doinstalovat dodatečně skrze aplikační kontejner Karaf podle toho, jaké funkcionality uživatel požaduje. K vlastnostem nacházejícím se v raném stádiu vývoje bude uveden pouze jejich popis. Kompletní výčet veškerých vlastností, které je možné doinstalovat lze nalézt v oficiální dokumentaci ODL kontroléru.[6]

Tabulka 2 – Vlastnosti dodávané společně s ODL kontrolérem

Vlastnost	Název**
Application-Layer Traffic Optimization	odl-alto-release
Authentication, Authorization and Accounting	odl-restconf
Centinel	odl-centinel-all
Controller shield	odl-usecplugin
Device Identification and Driver Management	odl-didm-all
Fabric as a Service	X
Group Based Policy	odl-groupbasedpolicy-ofoverlay, odl-groupbasedpolicyi-ui
L2 Přepínač*	odl-l2switch-switch-ui
Messaging For Transport	odl-mdsal-all, odl-messaging4transport-api, odl-messaging4transport
NetIDE	odl-netide-rest
Network Intent Composition	odl-nic-core-mdsal, odl-nic-console, odl-nic-listeners
Network Modeling	odl-nemo-openflow-renderer, a odl-nemo-engine-ui
Neutron & OVSDB	odl-ovsdb-openstack
NeXt	X
Service Function Chaining	odl-sfc-core, odl-sfc-ui, odl-sfc-netconf, odl-sfc-test-consumer, odl-sfc-ovs, odl-sfcofl2
Time Series Data Repository	odl-tsdr-hsqldb-all
Topology processing framework	odl-topoprocessing-framework
User network interface manager	odl-unimgr
Virtual Tenant Network	odl-vtn-manager-rest
VPN	odl-vpnservice-core
YANG PUBSUB	odl-yangpush-rest

*popsáno u Ryu kontroléru

**název/názvy sloužící k doinstalování dané vlastnosti skrze aplikační kontejner Karaf (některé vlastnosti se skládají z více částí), X znamená, že se vlastnost nachází ve stádiu vývoje

[6] https://www.opendaylight.org/opendaylight-features-list

ALTO (Application-Layer Traffic Optimization) je IETF (Internet Engineering Task Force) protokol, který poskytuje informace o síti přímo aplikacím, které běží na aplikační vrstvě referenčního modelu ISO/OSI. ALTO definuje abstrakce a služby, které poskytují zjednodušené pohledy na síť a služby. Umožňuje tak aplikacím optimálně využívat síťové zdroje (Internet Research Task Force, 2014). Služeb ALTO mohou využívat aplikace, které mají možnost volby, ke kterému koncovému bodu se připojí (např. peer-to-peer aplikace).

AAA (Authentication, Authorization and Accounting) umožňuje autorizaci, autentizaci a správu účtů založenou na Apache Shiro Java Security Framework. Autentizace je založena na poskytnutí uživatelského jména a hesla ODL kontroléru, který obratem poskytne tzv. token. Token slouží např. k přístupu k síťovým topologiím skrze *northbound* API. Autorizace je založena na tom, že každý účet má určitá práva. Například k modifikaci a čtení určitých topologií. Veškeré přístupy mohou být zaznamenány a použity například k analýze, diagnostice a bezpečnostnímu auditu. RESTCONF protokol, který slouží webovým aplikacím k přístupu ke konfiguračním a stavovým datům, je nejčastějším konzumentem AAA služeb. AAA služby jsou instalovány automaticky společně s protokolem RESTCONF.

Centinel poskytuje distribuovaný a spolehlivý framework sloužící k efektivnímu sběru, agregaci a „sinku“ (funkce nebo třída navržená tak, aby přijímala události z dalších objektů nebo funkcí) datových proudů napříč perzistentní databází a analyzátory. Umožňuje dalším vlastnostem přijímat události z několika datových proudů a provádění akcí (např. síťová konfigurace, dávkové zpracování).

Controller shield vytváří repositář, který se nazývá UsecPlugin (Unified-Security Plugin). Poskytuje bezpečnostní informace kontroléru *northbound* aplikacím. Umožňuje shromažďovat zdroje různých útoků zaznamenaných na *southbound* zásuvných modulech nebo shromažďovat informace o podezřelých či důvěryhodných kontrolérech nacházejících se v síti. Shromážděné informace mohou být použity ke konfiguraci firewallu a k vytvoření tzv. „černé listiny“ IP adres.

Device Identification and Driver Management poskytuje funkcionality specifické pro dané zařízení. Kód běžící na zařízení má díky této vlastnosti povědomí o možnostech a omezeních daného zařízení.

FaaS (Fabric as a Service) vytváří společnou abstraktní vrstvu nad fyzickou topologií sítě. Implementuje rozhraní, která by mohla v budoucnu poskytnout abstrakci nad tradičními *southbound* rozhraními, jako jsou OVSDB, OpenFlow a další. Řešení by bylo, na rozdíl od

OpenFlow, nezávislé na použitých zařízeních, výrobci a technologiích. V podstatě by byla přidána mezi síťové moduly uvnitř kontroléru a *southbound* rozhraní další vrstva. Díky této vrstvě by bylo možné se soustředit pouze na to, co má aplikace dělat, a nebylo by zapotřebí zabývat se detaily síťové implementace a konfiguračními příkazy.

GBP (Group Based Policy) umožňuje uživatelům vyjádřit konfiguraci sítě deklarativním způsobem, což znamená, že uživatel pouze specifikuje, co se má udělat a nikoli jak se to má udělat. Poskytuje *application-centric* model ODL kontroléru, který odděluje informace ohledně požadavků aplikace na připojení k síti od informací týkajících se detailů síťové infrastruktury. *Application-centric* model slouží ke správě celé sítě skrze aplikaci. Pomocí GBP lze rozdělit konkrétní koncové body do určitých skupin a stanovit pravidla komunikace mezi celými skupiny a mezi jednotlivými koncovými body. Například lze nastavit L3 směrování nebo velikost L2 *flood* domény. BGP lze u ODL kontroléru ovládat pomocí příkazů přes REST API nebo více interaktivně skrze webové uživatelské rozhraní.

L2 přepínač má oproti stejnojmennému modulu u Ryu kontroléru implementován protokol STP.

Messaging For Transport je tzv. MOM (Message-Oriented Middleware), který poskytuje vlastnostem vrstvu zvanou *middleware*. Tato vrstva umožňuje softwarovým komponentám, které byly vyvíjeny zvlášť a běží na různých platformách, vzájemnou komunikaci (Oracle Corporation and/or its affiliates, 2010). Vývojáři se mohou tedy soustředit pouze na to, jak bude jejich aplikace na přijaté zprávy reagovat.

NetIDE umožňuje přenositelnost a kooperaci v rámci jedné sítě, ve které se nachází více kontrolérů. Používá se k tomu architektura klient/server. Na klientských SDN kontrolérech běží různé SDN aplikace, které mají přístup k síti skrze abstrakci a koordinaci, kterou jim poskytuje jeden serverový SDN kontrolér (ODL). Umožňuje spouštět aplikace napsané pro Ryu, Floodlight či Pyretic kontrolér na síťové infrastruktuře, která je spravována ODL kontrolérem. Poskytuje také integrované vývojářské prostředí, které umožňuje vyvíjet a testovat aplikace. Toto prostředí obsahuje také grafické rozhraní pro specifikaci topologie sítě, uživatelské rozhraní pro nasazení konfigurace a editory sloužící ke specifikaci prostředí sítě pro simulaci.

NeMo (Network Modeling) je doménově specifický jazyk sloužící k abstrakci síťového modelu a identifikaci operačních vzorů. Umožňuje uživatelům nebo aplikacím sítě popsat požadavky na síťové zdroje, služby a logické operace intuitivním způsobem (co se má udělat

a ne jak). Princip NeMo se velmi podobá jazyku SQL. ODL poskytuje NeMo pro OpenFlow a klasická zařízení. NeMo poskytuje aplikacím *northbound* rozhraní, které propojuje aplikace a kontrolér. K výměně příkazů mezi NeMo a kontrolérem slouží REST API. NeMo je také integrován do grafického rozhraní kontroléru.

NIC (Network Intent Composition) poskytuje abstraktní vrstvu pro práci s vytvořenými záměry sítě. Umožňuje kontroléru spravovat a řídit síťové služby a prostředky na základě tzv. záměrů. Záměry je možné vytvářet skrze CLI ODL kontroléru. Pomocí vytvořených záměrů lze specifikovat chování sítě. Například lze zablokovat komunikaci mezi dvěma koncovými zařízení nebo nastavit kvalitu služeb.

Neutron & OVSDB je vlastnost, která se skládá z několika služeb a zásuvných modulů ODL kontroléru, které dohromady poskytují zjednodušenou integraci s OpenStack Neutron frameworkem. Tyto služby umožňují „OpenStacku“ snížení zátěže ODL kontroléru a současně umožňují ODL poskytovat rozšířené síťové služby „OpenStacku“. Slouží také k virtualizaci sítě. OpenStack je cloudový operační systém, který řídí velké množiny počítačů. OpenStack Neutron je SDN projekt zaměřený na poskytování sítě jako služby ve virtuálních výpočetních prostředích (SDNCentral, c2012–2017b).

NeXt poskytuje webové uživatelské rozhraní ke zobrazení a tvorbě topologie sítě. NeXt dokáže zobrazit velké komplexní topologie, agregované síťové uzly, tunely, skupiny. Je ho možné spárovat s ODL kontrolérem a zobrazit tak topologii, kterou ODL kontrolér spravuje.

SFC (Service Function Chaining) slouží k naprogramování určité posloupnosti akcí pro konkrétní paket. K doručení end-to-end služeb jsou často zapotřebí různé servisní funkce. Mezi tradiční síťové servisní funkce patří například firewally, NAT (Network Address Translation) a také funkce specifické pro určité aplikace. Definice a konkretizace uspořádané soustavy servisních funkcí a následné řízení síťového provozu dle soustavy se nazývá SFC. SFC lze u ODL kontroléru spravovat skrze grafické rozhraní. Uživatel může vytvářet servisní funkce, které poté může přiřadit do vytvořeného řetězce funkcí (Internet Research Task Force, 2015b).

TSDR (Time Series Data Repository) umožňuje sběr různých statistických dat z SDN zařízení a jejich uložení do datového uložiště. Uložená data mohou být použita v aplikacích postavených nad TSDR pro detekci bezpečnostních rizik, analýzu výkonu, optimalizaci konfigurace atd. K dispozici jsou dva typy uložišť. Uživatel si může vybrat, zda bude data

ukládat v NoSQL nebo MySQL databázi. TSDR poskytuje framework pro připojení požadovaných sběračů dat, pomocí nichž lze sbírat například OpenFlow statistiky a další.

Topology processing framework poskytuje filtrované a sloučené pohledy na síťové topologie.

User network interface manager umožňuje konfiguraci user network interface v síťových zařízeních a využívání služeb definovaných Metro Ethernet Fórem, jako je například Carrier Ethernet pro propojení síťových zařízení. User network interface je v podstatě hranice, která rozděluje zodpovědnost mezi zákazníkem a poskytovatelem internetu.

VPN (Virtual Private Network) implementuje služby infrastruktury, které jsou potřebné pro podporu služeb linkové a síťové VPN. L3 VPN služba v ODL kontroléru poskytuje framework, který slouží k vytvoření VPN založené na BGP-MP (Multi Border Gateway Protocol). BGP-MP umožňuje směrování více různých protokolů, jako je například L3 VPN.

VTN (Virtual Tenant Network) poskytuje *multi-tenant*[7] virtuální síť na SDN kontroléru. Poskytuje logickou abstraktní vrstvu, která umožňuje kompletní oddělení logické vrstvy od fyzické. Uživatelé si díky VTN mohou navrhnout a implementovat libovolnou síť bez znalosti fyzické topologie. Jakmile je síť navržena na VTN, je automaticky namapována na fyzickou topologii a poté konfigurována pomocí řídícího SDN protokolu. Vlastnost umožňuje lepší zacházení se síťovými zdroji, rychlejší změnu konfigurace a minimalizuje možnost vzniku konfiguračních chyb. Ke konstrukci virtuální sítě jsou používány virtuální uzly, rozhraní a spojení. Uživatel spravuje a vytváří virtuální síť pomocí REST API.

YANG PUBSUB umožňuje uživateli nastavit odběr konkrétních nastavení zařízení a způsob jejich odebírání. Uživatel může například stanovit, že bude odebírat nastavení určitého rozhraní každých deset vteřin nebo v momentě, kdy dojde ke změně konfigurace.

[7] *Multi-tenancy* je architektura, ve které jedna instance softwarové aplikace slouží několika zákazníkům. V této architektuře se každý zákazník nazývá *tenant*. Zákazníkům může být umožněna úprava některých částí aplikace, ale není jim dovoleno upravovat kód aplikace (Rouse, 2014).

3.5 Protokoly

V této podkapitole budou představeny některé z protokolů (tabulka 3), které jsou dodávány společně s ODL kontrolérem. Stejně jako vlastnosti, je i protokoly možné doinstalovat zvlášť skrze aplikační kontejner Karaf ve formě *southbound* zásuvných modulů. Kompletní seznam protokolů je možné najít na oficiálních stránkách ODL.[8]

Tabulka 3 – Protokoly dodávané společně s ODL kontrolérem

Protokol	Název**
Control And Provisioning of Wireless Access Points	odl-capwap-ac-rest
Border Gateway Protocol	odl-bgpcep-bgp
Internet of Things Data Management	odl-iotdm-onem2m
Link Aggregation Control Protocol*	odl-lacp-ui
Network Configuration Protocol	odl-netconf-mdsal, odl-netconf-connector, odl-netconf-topology, odl-restconf
OF-CONFIG	odl-of-config-all
Open vSwitch Database	odl-ovsdb-ui
Secure Network Bootstrapping Interface	odl-snbi-all
Simple Management Network Protocol	odl-snmp-plugin
Source Group Tag eXchange Protocol	odl-sxp-api, odl-sxp-core, odl-sxp-controller
The Locator/ID Separation Protocol	odl-lispflowmapping-msmr
Unified Secure Channel	odl-usc-channel-ui

*popsáno u Ryu kontroléru

**název/názvy sloužící k doinstalování daného protokolu skrze aplikační kontejner Karaf (některé protokoly se skládají z více částí)

CAPWAP (Control And Provisioning of Wireless Access Points) umožňuje jednomu kontroléru řídit kolekci přístupových bodů. Protokol je nezávislý na linkové vrstvě. Umožňuje vývoj inteligentních aplikací, které mají přístup k přístupovým bodům skrze REST API (Internet Research Task Force, 2009a).

BGP (Border Gateway Protocol) je protokol, který slouží ke směrování mezi autonomními systémy. Protokol lze konfigurovat skrze *northbound* REST API.

IoTDM (Internet of Things Data Management) umožňuje komunikaci zařízením, která spadají do internetu věcí. ODL kontrolér funguje jako *middleware,* přes který spolu zařízení komunikují. Autorizované aplikace díky tomu mohou získávat data z libovolných zařízení a mohou tato zařízení také řídit. Zásuvný modul obsahuje protokoly CoAP (Constrained Application Protocol), MQTT (MQ Telemetry Transport) a HTPP, které slouží ke komunikaci

[8] https://www.opendaylight.org/opendaylight-features-list

mezi zařízeními. Všechny tyto protokoly slouží ke komunikaci *machine-to-machine*, což může být libovolná technologie, která umožňuje výměnu informací a provádění akcí bez lidského zásahu (Rouse**,** 2010)

NETCONF (Network Configuration Protocol) protokol definuje jednoduchý mechanismus, jehož prostřednictvím lze spravovat síťová zařízení. Protokol poskytuje API, jehož prostřednictvím je aplikacím umožněno přijímat a odesílat konfigurační data. Protokol NETCONF využívá paradigmatu vzdáleného volání procedur (Internet Research Task Force, 2011). Vzdálené volání procedur je kódováno pomocí formátu XML (eXtensible Markup Language).

OF-Config slouží ke správě OpenFlow zařízení. Byl definován jako schéma NETCONF protokolu, umí provádět akvizice (získávání) stavu a nastavení logického přepínače, portu a fronty.

OVSDB (Open vSwitch Database) je protokol sloužící ke správě zařízení SDN sítí. Protokol byl vytvořen týmem Nicira, který později získala společnost VMware, Inc. Původně sloužil pro správu OVS (Open vSwitch), ale nyní tento protokol podporuje mnoho výrobců síťových zařízení. OVSDB lze použít v prostředí, kde je nasazen OpenFlow protokol k vlastní konfiguraci OVS zařízení (SDNCentral, c2012–2017c). V prostředí, kde není nasazen OpenFlow protokol slouží OVSDB ke kompletní konfiguraci zařízení (např. směrování).

SNBI (Secure Network Bootstrapping Interface) umožňuje automatické navázání spojení s připojeným SNBI zařízením, přidělení unikátní adresy a vytvoření zabezpečeného spojení. Díky automatické detekci nově připojených zařízení získává kontrolér celkový obraz fyzické topologie sítě a informace o typu připojených zařízení. Na základě těchto informací může kontrolér přiřazovat zařízení do určitých domén.

SNMP (Simple Network Management Protocol) umožňuje vlastnostem kontroléru konfigurovat zařízení. Jedná se o jednoduchý, standardizovaný protokol pro správu síťových zařízení. Díky tomuto protokolu může SDN kontrolér konfigurovat i zařízení, která nepodporují OpenFlow protokol. Dovoluje vlastnostem zastávat funkci SNMP manažera, který řídí a monitoruje síť pomocí příkazů, které posílá SNMP agentovi, který se nachází na zařízení. Tento agent je zodpovědný za provedení příkazů, které byly poslány manažerem.

Source Group Tag eXchange Protocol slouží k propagaci vazby mezi IP adresou a tzv. SGT (Source Group Tag). SGT je unikátní identifikátor skupiny koncových bodů, které sdílejí

stejná síťová pravidla. Skupinu, do které koncový bod patří, lze nastavit staticky či dynamicky. Díky unikátnímu identifikátoru lze na skupinu uplatňovat různá síťová pravidla. SGT má stejný význam jako skupina u GBP.

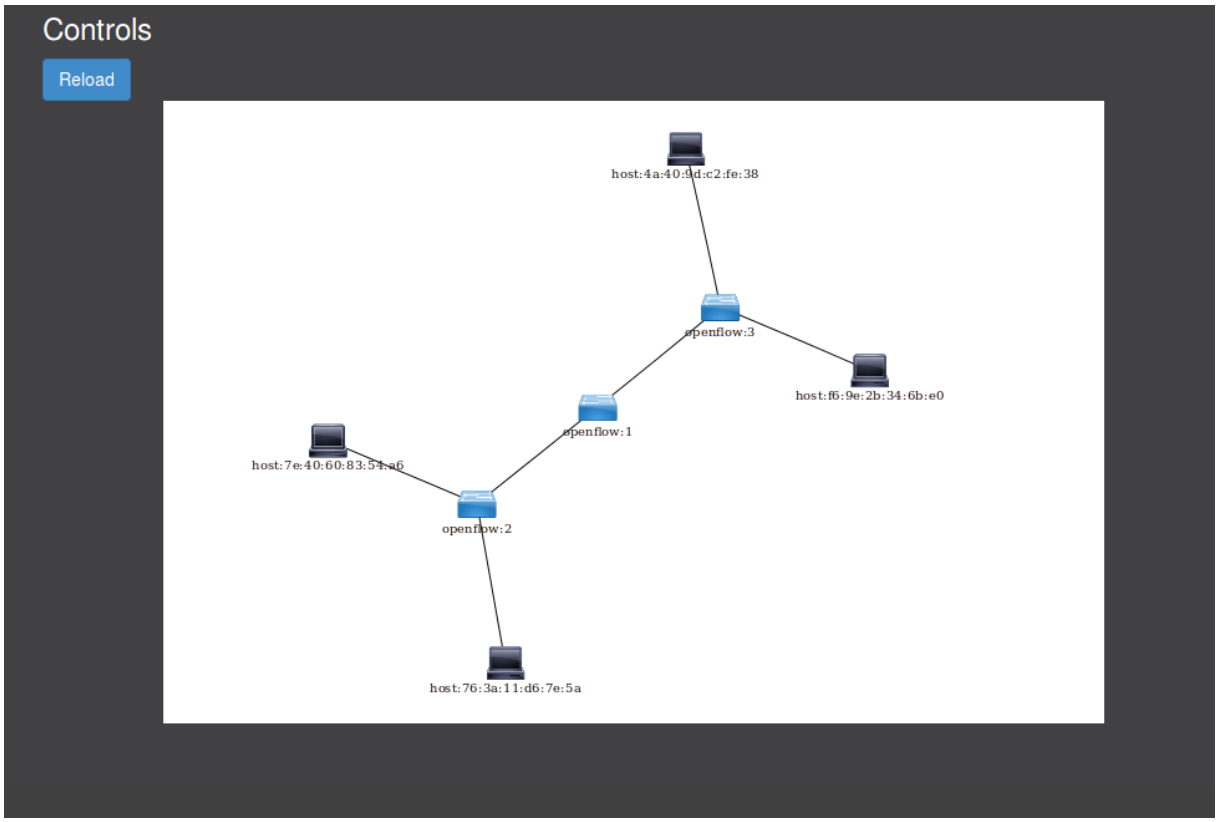
LISP (The Locator/ID Separation Protocol) je směrovací protokol vytvořený společností Cisco, který byl v roce 2013 uvolněn jako volný standard. LISP protokol od sebe odděluje lokační a identifikační funkci IP adresy. IP adresa je tedy rozdělena na dva oddělené prostory. EID (Identifikátor koncového bodu) sloužící k identifikaci koncového zařízení a RLOC (Routing Locator) ke zjištění, kde se dané zařízení nachází. RLOC se přiřazuje routerům. Pokud se EID cílového zařízení nenachází ve stejném prostoru, tak je paket odeslán na hraniční router a k vlastnímu směrování dochází pomocí RLOC (Internet Research Task Force, 2013).

USC (Unified Secure Channel) umožňuje vytvoření zabezpečeného kanálu mezi kontrolérem a zařízením. K zabezpečení kanálu lze použít protokol TLS (Transport Layer Security) nebo DTLS (Datagram Transport Layer Security). Zabezpečený kanál se využívá ke konfiguraci sítě a jednotlivých zařízení, díky zabezpečení nemůže být zařízení konfigurováno nedůvěryhodným kontrolérem. Architektura USC se skládá z USC agenta, USC zásuvného modulu, USC manažera a USC uživatelského rozhraní. Agent běží na zařízení a je zodpovědný za iniciování a udržování spojení s kontrolérem. *Southbound* USC zásuvný modul se nachází na kontroléru a je zodpovědný za zabezpečenou komunikaci mezi agentem a kontrolérem. Manažer provádí konfiguraci zařízení, zajišťuje vysokou dostupnost a monitorování. Uživatelské rozhraní USC je integrováno do webového grafického rozhraní ODL kontroléru.

3.6 Grafické rozhraní

DLUX (OpenDaylight User Experience) je grafické rozhraní ODL kontroléru (obrázek 8). Ke grafickému rozhraní se přistupuje přes webový prohlížeč skrze adresu *http://<ip adresa kontroléru>:8181/index.html#/login*. Po zadání této URL adresy do webového prohlížeče je vyžadováno uživatelské jméno a heslo. Heslo a jméno je ve výchozím nastavení „admin“. Grafické rozhraní umožňuje uživateli ovládání nainstalovaných aplikací a vizualizaci topologie včetně IP adres, unikátních identifikátorů a portů jednotlivých zařízení. Většina vestavěných aplikací ODL kontroléru, které stačí pouze doinstalovat skrze aplikační kontejner Karaf, umožňuje integraci do grafického rozhraní kontroléru. Aplikace může obsahovat vlastní grafické uživatelské rozhraní, které je přístupné ve formě záložky nebo ke svému

ovládání poskytuje pouze REST API, do kterého má uživatel přístup skrze „YANG UI“ záložku. Záložka „YANG UI“ umožňuje zobrazit veškerá poskytovaná REST API, přičemž konkrétní aplikaci je možné snadno vyhledat podle názvu. Uživatel vybere konkrétní REST API požadované aplikace a grafické rozhraní DLUX automaticky sestaví příkaz. Uživatel poté pouze pomocí vstupních textových polí příkaz konkretizuje (například doplní unikátní identifikátor zařízení) a odešle. Uživatelské rozhraní je třeba doinstalovat skrze aplikační kontejner Karaf pomocí příkazu `odl-dlux-all`.



Obrázek 8 – Ukázka vizualizace topologie v grafickém rozhraní kontroléru ODL

Uživatelské jméno a heslo je možné změnit, ale není možné dalšího uživatele přidat. Pro změnu přihlašovacích údajů se využívá REST API kontroléru ODL. Nejprve je zapotřebí vytvořit JSON soubor v odpovídajícím formátu. Pomocí tohoto souboru je poté uživatel aktualizován. S administrátorskými právy je možné skrze rozhraní REST získat výpis všech uživatelů. Na rozdíl od kontroléru ONOS, však není možné získat jejich hesla. ODL má oproti kontroléru ONOS ještě další výhodu. Přihlašovací údaje lze u ODL kontroléru měnit za běhu kontroléru a není zapotřebí jeho restart. Kompletní návod na změnu uživatelského jména a hesla lze nalézt na oficiálních stránkách kontroléru ODL.[9]

[9] https://wiki.opendaylight.org/view/AAA:Changing_Account_Passwords

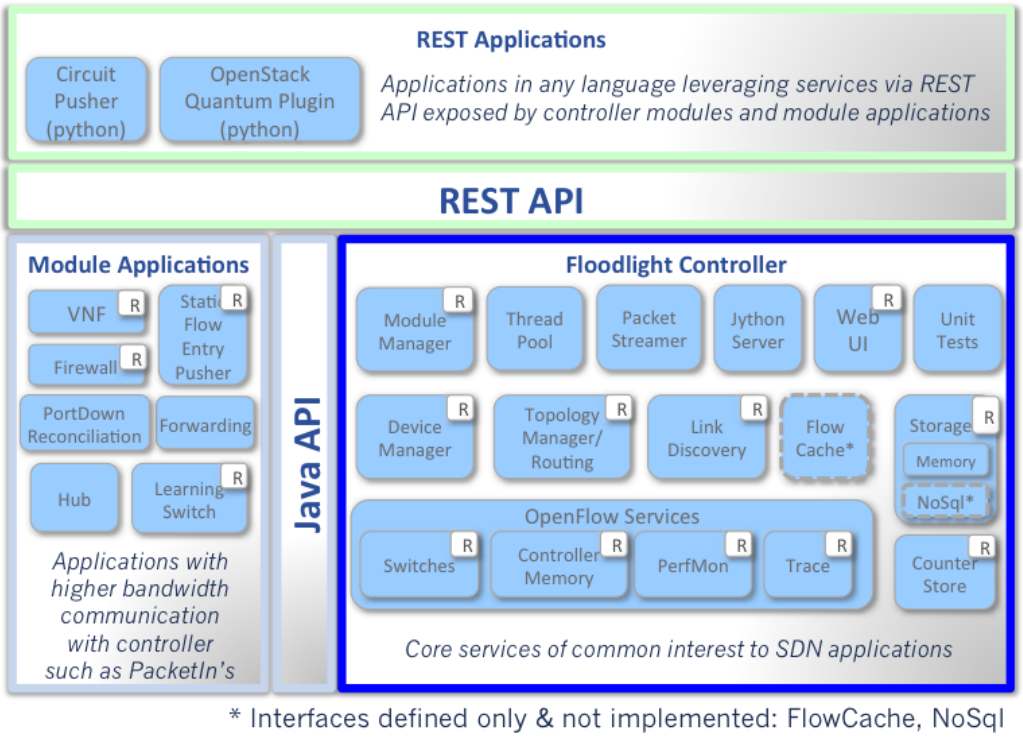
4 FLOODLIGHT

Kapitola, pokud není uvedeno jinak, čerpá z oficiální dokumentace kontroléru Floodlight (Atlassian, c2017).

Floodlight je otevřený OpenFlow kontrolér softwarově definovaných sítí založený na Javě a licencovaný Apachem. Jako základ Floodlight kontroléru byl použit otevřený Beacon SDN kontrolér vyvinutý Stanfordskou univerzitou. Nicméně Floodlight kontrolér se v porovnání s Beacon kontrolérem velice rozrostl a stal se jedním z nejpopulárnějších otevřených SDN kontrolérů. Při spuštění Floodlight kontroléru dojde, stejně jako u dalších kontroléru, k aktivaci *northbound* a *southbound* rozhraní. To znamená, že dojde ke spuštění kontroléru a sady nakonfigurovaných aplikačních modulů. Všechny běžící moduly poskytují REST API, která jsou ve výchozím nastavení dostupná na portu 8080. Tento port je možno přenastavit, nebo lze používat i HTTPS (Hypertext Transfer Protocol Secure). Jakákoli aplikace může komunikovat s kontrolérem pomocí REST příkazů. Na druhé straně je poskytováno *southbound* rozhraní pro SDN zařízení, kde se nachází protokol OpenFlow. Kontrolér naslouchá na specifickém OpenFlow portu. Pomocí tohoto portu jsou vytvářena spojení mezi Floodlight kontrolérem a OpenFlow zařízeními. Díky rozšiřitelnému vývojovému prostředí Javy a jádrem vyvinutým společností Big Switch Network se jedná o robustní a pro uživatele snadno použitelný kontrolér (Sridhar, 2015a; Nadeau & Gray, c2013, s. 93–95; Project Floodlight, c2017).

4.1 Architektura

Floodlight není pouze OpenFlow kontrolér, ve skutečnosti se jedná o kontrolér a kolekci aplikací, které jsou postaveny na Floodlight kontroléru. Floodlight kontrolér realizuje sadu běžných funkcionalit sloužících k řízení a dotazování se OpenFlow sítě, zatímco aplikace nad těmito běžnými funkcionalitami realizují různé funkce, díky nimž síť funguje dle požadavků uživatele. Obrázek 9 ukazuje vztah mezi Floodlight kontrolérem, vestavěnými Java aplikačními moduly kompilovanými pomocí Floodlight a aplikacemi postavenými nad Floodlight REST API (Sridhar, 2015a; Nadeau & Gray, c2013, s. 93–95).



Obrázek 9 – Architektura Floodlight kontroléru

Zdroj: (Atlassian, c2017)

Architektura kontroléru je modulární. Obsahuje komponenty pro správu topologie, správu zařízení, výpočet cesty, infrastrukturu pro webový přístup, uložiště počítadel a zobecněnou abstrakci úložiště sloužící k ukládání stavů. S těmito komponentami je zacházeno jako se zásuvnými službami s rozhraním, které exportuje stav. Kontrolér prezentuje sadu rozšiřitelných REST API a také notifikační událostní systém. API umožňuje aplikacím získat a nastavit stav kontroléru, stejně jako přihlášení k odběru událostí vyvolaných kontrolérem pomocí *Java Event Listener*.

Floodlight zahrnuje širokou škálu služeb, od zjištění stavů sítě až po události umožňující komunikaci přepínačů s utilitami, jakou jsou úložiště, vlákna a webová uživatelské rozhraní.

Základní modul nazývaný *Floodlight Provider* zpracovává vstupní a výstupní operace z přepínačů a překládá OpenFlow zprávy na události, které mohou být zpracovány dalšími moduly. Floodlight zahrnuje vláknový model, který umožňuje sdílení vláken mezi moduly. Sdílená data jsou chráněna proti znehodnocení pomocí synchronizačních zámků, které zaručují výlučný přístup. Závislosti mezi komponentami jsou vyřešeny během spouštění kontroléru skrze konfiguraci.

Modul *device manager* udržuje aktuální informace o zařízeních nebo koncových bodech skrze *packet-in* zprávy. Díky informacím obsažených v *packet-in* se může kontrolér naučit, ke kterému portu na určitém přepínači je koncové zařízení připojeno.

Link Discovery a *topology manager* používá ke zjištění zařízení a spojení mezi nimi protokol LLDP.

Packet streamer slouží k předání OpenFlow paketů jakémukoli monitorovacímu zařízení, které je připojeno k síti. Poskytuje rozhraní pro specifikaci OpenFlow zprávy, která bude předávána. Typicky je označován jako filtr.

Memory storage source slouží k ukládání sdílených dat. Floodlight moduly, které na tom závisí, mohou vytvářet, modifikovat a mazat data. Kromě toho se také moduly mohou registrovat ke změně dat.

ThreadPool slouží k zabalení Floodlight modulů pro Java *ScheduledExecutorService*. Používá se k tvorbě vláken v určitém čase nebo periodě.

4.2 Instalace

Floodlight kontrolér je založený na Javě. K buildování je zapotřebí mít nainstalovaný Apache Ant[10] nebo Maven a k jeho spuštění je zapotřebí mít nainstalovaný také balíček JDK, který obsahuje JVM. Floodlight je také možné spouštět ve vývojovém prostředí Eclipse. Před vlastní instalací je ještě zapotřebí nainstalovat balíček Python development. Obsahuje hlavičkové soubory pro Python C API a vše potřebné ke kompilaci rozšiřujících python modulů (Python Software Foundation, c1990–2017). Pro správnou a optimální funkčnost by měla být nastavena proměnná prostředí `JAVA_HOME`.

Stejně jako Ryu má i Floodlight nízké nároky na operační paměť. Pro virtuální stroj stačí vyhradit 256 MiB operační paměti. Floodlight získává kompletní přehled o topologii s takto malou operační pamětí již za 20 vteřin. Floodlight kontrolér je možné nainstalovat na libovolný operační systém. Oficiální dokumentace však doporučuje využít nejnovější LTS distribuci operačního systému Linux. Níže je uveden návod k instalaci Floodlight kontroléru na distribuci Ubuntu (16.04).

[10] Apache Ant je Java knihovna a nástroj příkazového řádků, jehož úkolem je řídit procesy popsané v souborech sloužících k buildu jako cíle a body rozšíření, které jsou navzájem na sobě závislé. Ant slouží hlavně k buildování (sestavování) Java aplikací. Ant dodává řadu vestavěných úkolů, které umožňují kompilovat, sestavovat, testovat a spouštět Java aplikace. Ant může být použit i k buildování aplikací, které nejsou naprogramovány v jazyce Java, například v jazyce C nebo C++. Obecně může být Ant použit k řízení jakéhokoli typu procesu, který lze popsat z hlediska cílů a úkolů (Apache Software Foundation, c2017).

Příkaz sloužící k instalaci softwaru k buildu (Ant) a balíčku python development:

```
#sudo apt-get install ant python-dev
```

Příkaz pro instalaci programu git:

```
#sudo apt install git
```

Příkaz pro zkopírování nejnovější a otestované verze Floodlight kontroléru:

```
#git clone git://github.com/floodlight/floodlight.git
```

Příkaz pro inicializaci submodelu (ve složce floodlight):

```
#git submodule init
```

Příkaz pro aktualizaci submodelu (ve složce floodlight):

```
#git submodule update
```

Příkaz pro rebuild kontroléru Floodlight (ve složce floodlight):

```
#ant
```

Příkaz pro vytvoření složky `/var/lib/floodlight`

```
#sudo mkdir /var/lib/floodlight
```

Příkaz pro nastavení práv adresáře `/var/lib/floodlight`, nastavení *read*, *execute*, *write* pro vlastníka, skupinu a ostatní:

```
#sudo chmod 777 /var/lib/Floodlight
```

4.3 Spouštění

U Floodlight kontroléru lze nakonfigurovat, které moduly budou kontrolérem zkompilovány a načteny. Uživatel může konfigurovat kontrolér skrze dva konfigurační soubory. Konfigurace je načítána vždy při startu kontroléru. K přenastavení konfigurace je nutné upravit konfigurační soubor a poté kontrolér restartovat. Tvůrci do budoucna plánují podporu změny konfigurace přímo za běhu kontroléru.

V souboru `net.floodlightcontroller.core.module.IFloodlightModule` se nachází veškeré zkompilované moduly. Všechny tyto moduly nicméně nemusí být načítány při startu kontroléru.

Ve výchozím nastavení se vybrané moduly, které budou po startu kontroléru načteny a spuštěny, nachází v souboru `src/main/resources/floodlightdefault.properties`. Uživatel může však použít i vlastní konfigurační soubor. Kromě nastavení modulů načítaných při startu kontroléru, lze v tomto souboru také nastavit různé konfigurační parametry. Mimo jiné port pro přístup k webovému uživatelskému rozhraní, časové limity a OpenFlow port pro spojení se zařízeními. Ke spuštění modulu či aplikace stačí upravit soubor `floodlightdefault.properties`. Níže jsou uvedeny různé varianty příkazů pro spuštění kontroléru Floodlight. Uživatel se musí nacházet ve složce, kde je Floodlight kontrolér nainstalován.

Příkaz pro spuštění Floodlight kontroléru s výchozím konfiguračním souborem:

```
#java -jar target/floodlight.jar
```

Příkaz pro spuštění kontroléru s vlastním konfiguračním souborem:

```
#java -jar target/floodlight.jar -cf /$CESTA/$NAZEV_KONFIGURACNIHO_SOUBORU
```

Příkaz sloužící ke specifikaci vlastnosti uživatelem zvoleného modulu, překrývá nastavení v konfiguračním souboru:

```
#java -Dnet.floodlightcontroller.$NAZEV_MODULU.$VLASTNOT=$HODNOTA -jar
target/floodlight.jar
```

4.4 Moduly

V této podkapitole budou představeny moduly dodávané společně s kontrolérem Floodlight. Oficiální Floodlight dokumentace nicméně neobsahuje jejich kompletní seznam. Ten je možné získat až po instalaci kontroléru. Je však nutné prohledat celý jeho adresář. Většina modulů se nachází v adresáři `/floodlight/target/bin/net/floodlightcontroller/`. Společně s kontrolérem je také dodávaná aplikace *Circuit pusher* sloužící k vytvoření obousměrné okruhu mezi dvěma koncovými zařízení.[11]

[11] https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343496/Circuit+Pusher

Tabulka 4 – Moduly dodávané společně s Floodlight kontrolérem

Modul	Název***
Access Control List*#	net.floodlightcontroller.accesscontrollist.ACL
Debug counter#	net.floodlightcontroller.debugcounter.DebugCounterServiceImpl
DHCP server#	net.floodlightcontroller.dhcpserver.DHCPServer, net.floodlightcontroller.dhcpserver.DHCPSwitchFlowSetter
Firewall	net.floodlightcontroller.firewall.Firewall
Forwarding#	net.floodlightcontroller.forwarding.Forwarding
Hub	net.floodlightcontroller.hub.Hub
L2 přepínač*	net.floodlightcontroller.learningswitch.LearningSwitch
Load Balancer#	net.floodlightcontroller.loadbalancer.LoadBalancer
OpenStack Neutron**	
Packet-In Processing Time#	
Port Down Reconciliation	net.floodlightcontroller.flowcache.PortDownReconciliation
Routing manager#	
Simple Fault Tolerance#	net.floodlightcontroller.simpleft.FT
Statistics Collector#	net.floodlightcontroller.statistics.StatisticsCollector
Static Flow Entry Pusher#	net.floodlightcontroller.staticentry.StaticEntryPusher
Topology manager#	
Virtual Network Filter	net.floodlightcontroller.virtualnetwork.VirtualNetworkFilter

*popsáno u Ryu kontroléru

**popsáno u ODL kontroléru

***název/názvy modulu pod kterým ho lze nalézt nebo přidat do konfiguračního souboru

#modul načítaný ve výchozím nastavení při každém startu kontroléru

Debug counter umožňuje registrovat, inkrementovat a vyčistit čítače. Umožňuje například zjistit, kolik *packet-in* události přišlo z konkrétního zařízení.

DHCP (Dynamic Host Configuration Protocol) server slouží k dynamickému přidělování IP adres, výchozí brány a DNS (Domain Name System) serveru koncovým zařízením. Tento modul zachycuje veškeré DHCP požadavky z připojených koncových zařízení a stará se o odpovědi. Používá DHCPPool ke správě veškerých IP adres, které „pronajímá“ koncovým zařízením.

Firewall se velice podobá ACL modulu. Nicméně modul firewall umožňuje detailnější nastavení pravidel. Například umožňuje nastavit pravidlo pouze konkrétnímu zařízení, hledání shody dle MAC adres a nastavení konkrétní priority pro dané pravidlo. Ve výchozím nastavení je modul načítán při každém startu kontroléru, avšak po startu je ho zapotřebí ještě zapnout skrze rozhraní REST nebo přidáním jakéhokoli pravidla pomocí webového uživatelského rozhraní. Povolena je pouze komunikace, která byla specifikována pomocí pravidel, ostatní komunikace je blokována.

Forwarding modul implementuje funkci L2 přepínače. Vkládá flow záznamy do jednotlivých OpenFlow zařízení a pomocí těchto záznamů jsou poté pakety směrovány od zdroje k cíli. Modul umožňuje nastavení kritérií shody. Podporuje shodu na základě VLAN ID, zdrojové/cílové MAC adresy, zdrojové/cílové IP adresy a zdrojového/cílového portu. K odstranění smyček v síti je použit, místo STP protokolu, Dijkstrův algoritmus nejkratší cesty. Modul není kompatibilní s žádným dalším modulem sloužící ke směrování paketů (např. L2 přepínač).

Hub modul je implementací rozbočovače. Rozbočovač je aktivní prvek počítačové sítě, který komunikaci přijatou na jednom portu zkopíruje a odešle všemi dostupnými porty. Po aktivaci tohoto modulu pomocí konfiguračního souboru se všechna OpenFlow zařízení v síti budou chovat jako rozbočovače. Modul není kompatibilní s žádným dalším modulem sloužící ke směrování paketů (např. L2 přepínač).

L2 přepínač poskytuje naprosto totožné funkcionality jako stejnojmenný modul Ryu kontroléru. Není kompatibilní s žádným dalším modulem sloužící ke směrování paketů (např. Forwarding).

Load Balancer poskytuje jednoduché vyvažování zátěže pro protokoly ICMP (Internet Control Message Protocol), TCP (Transmission Control Protocol) a UDP (User Datagram Protocol). K vyvažování zátěže mezi různými IP adresami se používá algoritmus round-robin na základě počtu připojení.

Packet-In Processing Time slouží k měření výkonu jednotlivých modulů.

Port Down Reconciliation slouží k aktualizaci flow tabulek na jednotlivých přepínačích v případě výpadku některého z portů. Pokud dojde k výpadku portu, tak kontrolér dostane od přepínače zprávu. *Port Down Reconciliation* modul vyhledá vyřazený port a následně smaže všechny flow záznamy, jejichž akce směruje paket na vyřazený port.

Routing manager slouží k zjišťování dostupných cest mezi koncovými body. Poskytuje REST API, které uživateli umožňuje získávání informací o jednotlivých cestách a nastavení jejich metriky pro stanovení ceny dané cesty.

Simple Fault Tolerance umožňuje nastavení záložní řídící vrstvy kontroléru. Kompletní konfiguraci modulu je možné nalézt na oficiálních stránkách kontroléru Floodlight.[12]

Statistics Collector slouží uživateli ke sbírání statistik z OpenFlow přepínačů. Pomocí tohoto modulu může uživatel například zjistit využívanou šířku pásma na jednotlivých přepínačích nebo konkrétním portu. *Statistics Collector* poskytuje REST API, pomocí kterého si uživatel může konkrétní statistiky zobrazit nebo je použít v jiném modulu. Při používání sesbíraných statistik je třeba vzít na vědomí zpoždění, které vzniká při jejich doručování kontroléru. Statistics Collector se v pevně stanoveném intervalu (tento interval je možné nastavit) dotazuje přepínačů, které na jeho dotazy odpovídají. Modul je zapotřebí aktivovat pomocí REST API.

Static Flow Entry Pusher umožňuje, na rozdíl od modulu *Forwarding*, proaktivní přístup ke směrování paketů. Proaktivním přístupem je myšleno vkládání flow záznamů do flow tabulek přepínačů pomocí kontroléru ještě před tím, než na přepínač dorazí pakety, které vykazují s těmito záznamy shodu. V takovém případě již přepínač nemusí kontaktovat kontrolér kvůli vyhodnocení daného paketu. *Static Flow Entry Pusher* poskytuje REST API, které uživateli umožňuje vkládat flow záznamy, vytvářet skupiny, mazat záznamy (konkrétní nebo všechny najednou) a zobrazovat vložené flow záznamy (pouze pomocí tohoto modulu). Uživatel může k ovládání modulu použít kromě curl příkazů[13] i webové grafické rozhraní, kde lze *Static Flow Entry Pusher* nalézt pod tlačítkem „Edit Static Flow Entries“. Skrze webové grafické rozhraní není možné použít veškeré funkce, které tento modul nabízí.

Topology manager slouží ke správě topologie na základě zjištěných spojení a připojených přepínačů.

Virtual Network Filter slouží k virtualizaci sítě na L2 vrstvě. Umožňuje vytvořit několik logických L2 domén na jedné fyzické L2 doméně. Izolace mezi jednotlivými logickými doménami se týká pouze *unicast* komunikace. Po startu kontroléru s načteným *Virtual Network Filter* modulem zde neexistují žádné virtuální sítě, což znemožňuje jakoukoli komunikaci. Flow tabulka přepínače zpočátku obsahuje pouze jeden záznam, který všechny pakety posílá kontroléru ve formě *packet-in* událostí. V kontroléru jsou veškeré *packet-in*

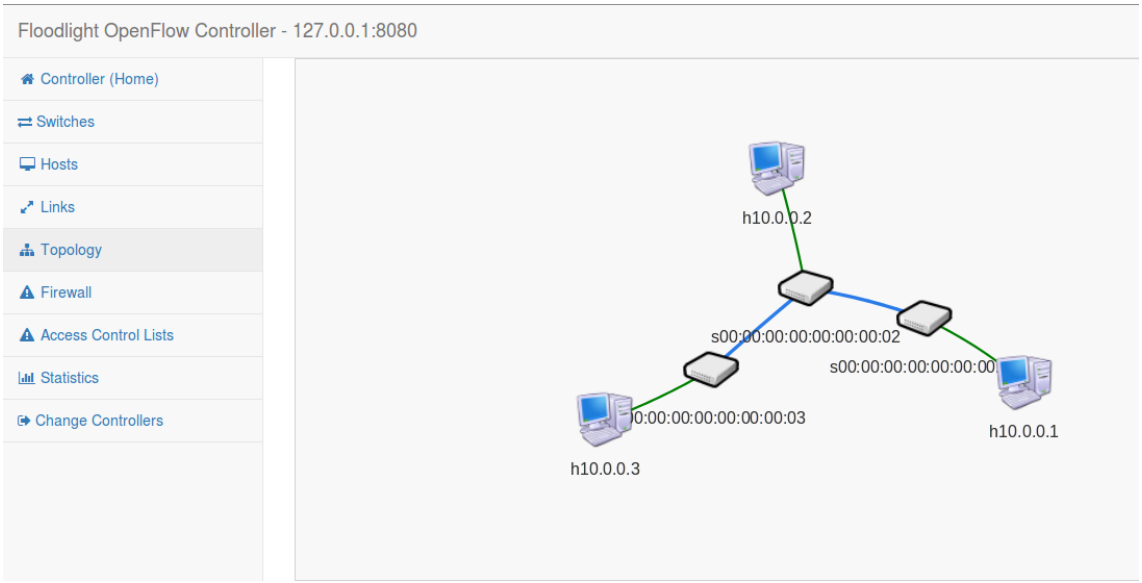
12 https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/36143107/How+to+Add+Fault+Tolerance+to+the+Control+Plane

13 https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343518/Static+Entry+Pusher+API#StaticEntryPusherAPI-AddingaGroup

události zpracovány nejprve *Virtual Network Filter* modulem. Modul analyzuje MAC adresu cíle a zdroje. Pokud se obě adresy nachází ve stejné virtuální síti, tak může být paket dále zpracováván například *Forwarding* modulem.

4.5 Grafické rozhraní

K webovému uživatelskému rozhraní (obrázek 10) se přistupuje pomocí webového prohlížeče skrze adresu *http://<ip adresa kontroléru>:8080/ui/pages/index.html*. Na rozdíl od kontrolérů ONOS a ODL není přístup k webovému uživatelskému rozhraní nijak zabezpečen, není tedy zapotřebí zadávat uživatelské jméno a heslo. Grafické rozhraní umožňuje uživateli ovládání některých nainstalovaných modulů, vizualizaci topologie včetně IP adres koncových zařízení a unikátních identifikátorů přepínačů, zobrazení různých OpenFlow statistik a informací o konkrétním přepínači, zobrazení informací týkajících se stavu kontroléru (doba běhu, spotřeba paměti RAM, nainstalované moduly, počet připojených přepínačů).



Obrázek 10 – Ukázka vizualizace topologie v grafickém rozhraní kontroléru Floodlight

5 OPEN NETWORKING OPERATING SYSTEM

Kapitola, pokud není uvedeno jinak, čerpá z komunitních webových stránek kontroléru ONOS (Onos Community, 2017).

ONOS (Open Networking Operating System) je škálovatelná a distribuovaná platforma SDN kontroléru, zaměřená na sítě poskytovatelů služeb a jejich požadavky. ONOS je vyvinutý neziskovou organizací ON.Lab[14], vytvořenou Berkeleyskou a Stanfordovou univerzitou. Cílem vývojářů ONOS je vytvořit takovou řídící platformu, která bude splňovat požadavky na dostupnost, rozšiřitelnost a vysoký výkon. ONOS se také zaměřuje na podporu více protokolů na *southbound* rozhraní, aby bylo možné komunikovat s různými zařízení a na poskytování správných API na *northbound* rozhraní tak, aby to vyhovovalo potřebám poskytovatelů služeb a vývojářům aplikací (Sridhar, 2015b).

ONOS může běžet jako distribuovaný systém napříč více servery. Umožňuje využívat procesory a paměťové zdroje více serverů a zároveň poskytuje odolnosti vůči chybám. Při pádu jednoho serveru díky tomu nedojde k narušení konektivity. Poskytuje podporu ke změně hardwaru či aktualizaci softwaru za běhu, aniž by byla narušena konektivita.

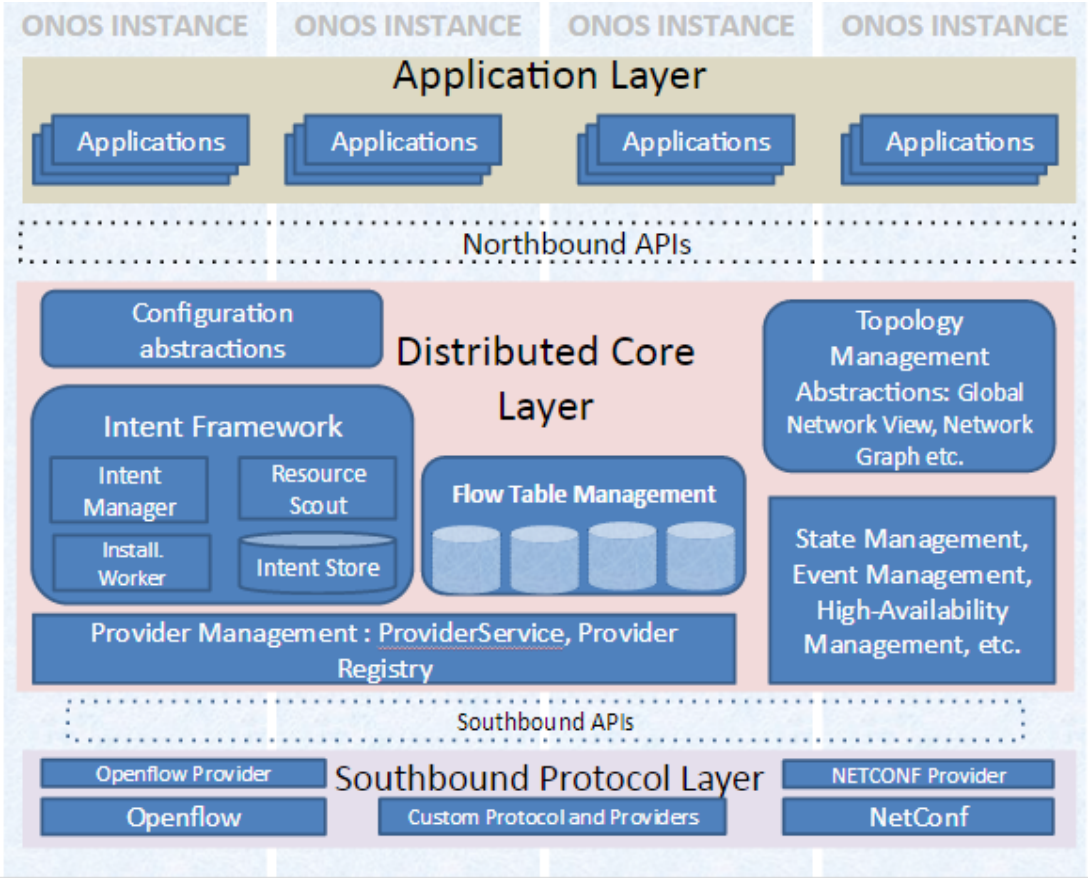
Jádro ONOS kontroléru a základní služby jsou stejně jako aplikace napsány v jazyce Java. Aplikace jsou ve formě balíčků načítány do aplikačního kontejneru Karaf, který je postaven na vrcholu OSGi frameworku. OSGi je modulární systém pro programovací jazyk Java, který umožňuje instalaci modulů a jejich dynamické spuštění v jednom JVM.

[14] ON.Lab se zabývá tvorbou open source nástrojů a platforem pro SDN

5.1 Architektura

Architektura ONOS kontroléru je rozdělena do tří vrstev (obrázek 11):

- Nejnižší vrstvy složené z modulů, které komunikují přímo se síťovými zařízeními a mají povědomí o konkrétním protokolu.
- Distribuované vrstvy jádra, která je nezávislá na konkrétním protokolu. Sleduje a podává informace o stavu sítě.
- Aplikační vrstvy skládající se z aplikací, jež mají ucelený pohled na topologii sítě a mohou ji efektivně řídit.



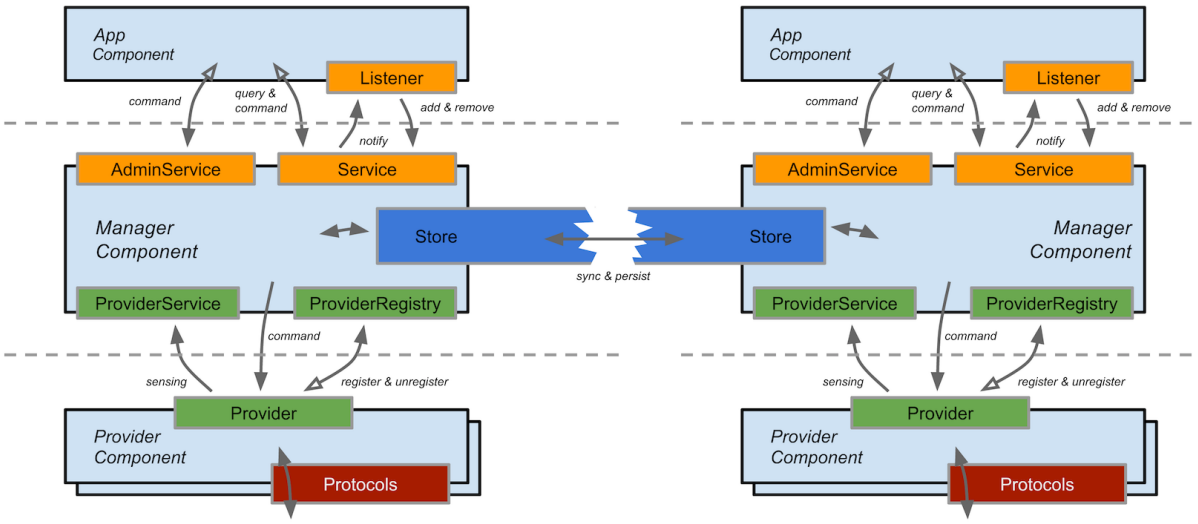
Obrázek 11 – Architektura kontroléru ONOS

Zdroj: (Sridhar, 2015b)

Na architekturu lze nahlížet také způsobem, že každá z vrstev představuje kolekci několika subsystémů, kde každý subsystém realizuje službu a je složen z několika komponent. Komponenty vytvářejí vertikální řez skrze vrstvy jako softwarový zásobník. Termíny služba a subsystém jsou u kontroléru ONOS zaměnitelné. Mezi subsystémy patří například *Cluster*, *Device*, *Packet*, *FlowRule*, *Path*, *Link*, *Host* a *Intent*.

5.1.1 Struktura subsystému

Každá z komponent subsystému je umístěna v jedné z hlavních vrstev a může být identifikována jedním nebo více Java rozhraními, které implementuje. Obrázek 12 shrnuje vztahy mezi jednotlivými komponentami subsystému. Čárkované čáry zobrazují hranice mezi vrstvami, které jsou vytvářeny *nortbound* a *southbound* rozhraním.



Obrázek 12 – Struktura subsystému

Zdroj: (Onos Community, 2017)

Provider je nejnižší vrstva zásobníku ONOS. *Provideři* komunikují se sítí pomocí knihoven specifických pro daný komunikační protokol a s jádrem pomocí rozhraní *ProviderService*. *Provideři* mají povědomí o konkrétním protokolu, jsou zodpovědní za interakci se sítí pomocí různých konfiguračních a řídících protokolů a za poskytování sesbíraných dat specifických pro danou službu jádru. *Provideři* mohou také sbírat data od dalších subsystémů a konvertovat je na data, která jsou specifická pro danou službu.

Provider je spojen s tzv. *ProviderID*. *ProviderID* umožňuje zařízením a dalším modelovým entitám zůstat ve spojení s daným poskytovatelem, který je odpovědný za jejich existenci i poté, co je poskytovatel odinstalován nebo uvolněn.

Subsystém může být spojen s několika *providery*. V takovém případě je *provider* stanoven buď jako primární nebo pomocný. Primární *provider* vlastní entity, které jsou spojeny s jeho službou. Zatímco pomocní *provideři* přispívají svými informacemi jako překryvy. Tento přístup dává přednost hlavnímu *providerovi*, pokud by překrytí mohlo mít za následek konflikt s podkladovými informacemi. *Device subsystem* je příkladem služby, která podporuje více *providerů*.

Manager je komponenta, která se nachází v jádře. *Manager* přijímá informace od *providerů* a posílá je aplikacím a dalším službám. Poskytuje několik rozhraní. *Northbound Service* rozhraní pomocí kterého se aplikace nebo další komponenty jádra mohou dozvědět o určitém aspektu stavu sítě. Rozhraní *AdminService* sloužící k přijímání administrativních příkazů a jejich aplikaci na stav sítě nebo systému. Southbound *ProviderRegistry* rozhraní přes které se mohou *provideři* registrovat u *managera*, aby s ním mohli komunikovat. Southbound *ProviderService* rozhraní poskytované registrovanému *providerovi*. Skrze toto rozhraní *provider* posílá a přijímá informace od *managera*. *Spotřebitelé* služby *Service* rozhraní mohou přijímat informace synchronně pomocí dotazování služby nebo asynchronně naslouchání události.

Úložiště se nachází v jádře a je úzce spojeno s *managerem*. *Úložiště* mají za úkol indexování, uchovávání a synchronizaci informací, které byly přijaty *managerem*. Tyto úkoly zajišťují konzistenci a robustnost informací napříč více ONOS instancemi pomocí přímé komunikace s *úložišti*, která se nachází v dalších instancích.

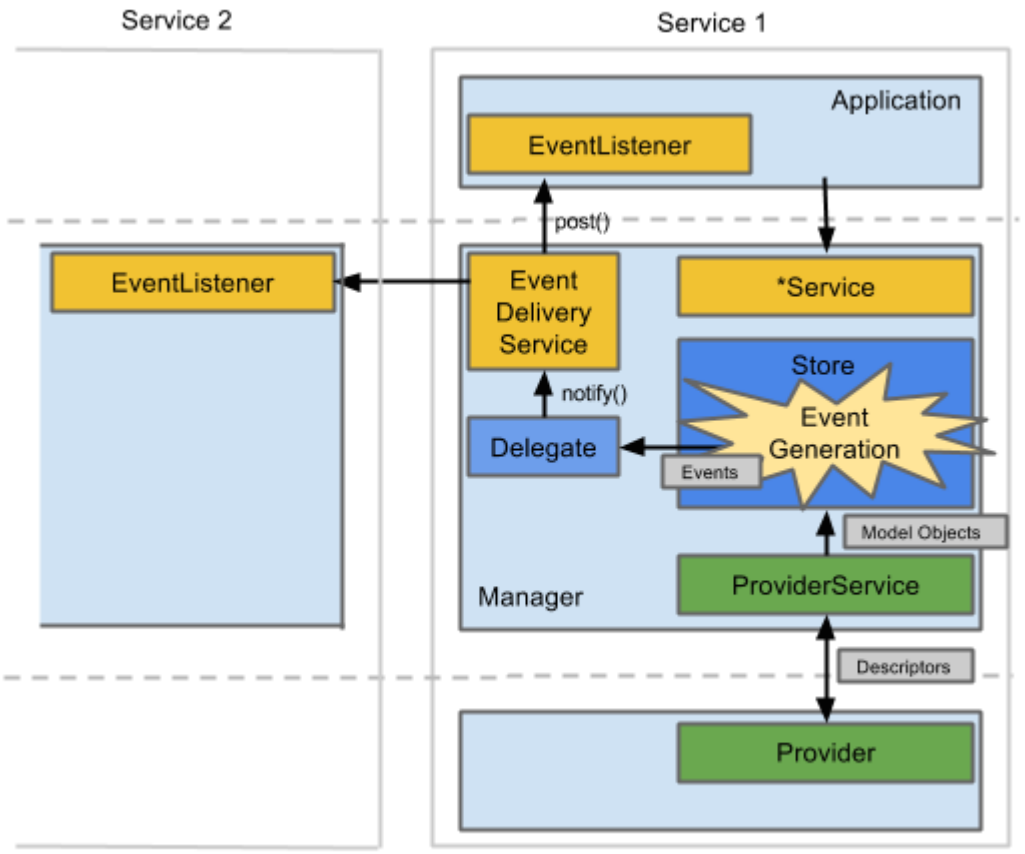
Aplikace spotřebovávají a manipulují s informacemi shromážděnými *managery* skrze *AdminService* a *Service* rozhraní. Aplikace mají širokou škálu funkcí od zobrazování topologie sítě ve webovém prohlížeči až po nastavení cest pro směrování.

Aplikace se registrují pomocí jména, které by mělo být ve tvaru obrácené DNS (Domain Name System) notace, např. *org.onlab.onos.fwd*, u tzv. *CoreService*. *CoreService* obratem poskytne aplikaci unikátní *ApplicationId*. Tento unikátní identifikátor používá ONOS ke sledování kontextu spojeného s aplikací (např. flow pravidla vytvořená danou aplikací).

5.1.2 Události a popisy

Dvě základní jednotky distribuce informací v rámci ONOS jsou události a popisy. Stejně jako služby, jsou události a popisy spojeny s konkrétními síťovými prvky a pojmy. Obě jsou po vytvoření neměnné.

Obrázek 13 rozvádí předchozí obrázek 12 týkající se struktury subsystému a ukazuje vztahy mezi událostmi, popisy a komponentami, které jsou popsány níže.



Obrázek 13 – Vztahy mezi událostmi a popisy

Zdroj: (Onos Community, 2017)

Popisy jsou používány k předávání informací o prvku přes *southbound* API. Například *HostDescription* obsahuje informace o hostově IP a MAC adrese a jeho poloze v rámci sítě (VLAN ID, zařízení/port). Popisy se obvykle skládají z jednoho nebo více modelových objektů a reprezentují v ONOS kontroléru různé komponenty sítě.

Události jsou používány *managery* k upozornění aplikací, které naslouchají událostem, o změnách v síti a *úložištěmi* k oznámení změn dalším instancím. Událost se skládá z typu události a předmětu tvořeného modelovými objekty. Například, *DeviceEvent* může být použit k upozornění *DeviceListeners*, že zařízení bylo detekováno, ztraceno nebo se změnil některý z jeho aspektů.

Události jsou generovány *úložištěm* na základě informací poskytnutých *managerem*. Jakmile je vygenerována událost, tak je odeslána pouze aplikacím, které této události naslouchají pomocí *StoreDelegate* rozhraní, což v konečném důsledku vyvolá *EventDeliveryService*. V podstatě *StoreDelegate* posouvá událost mimo úložiště a *EventDeliveryService* zajistí, že se

událost dostane pouze k aplikacím, které takové události naslouchají. Tyto složky se nachází v *manageru*, který poskytuje implementační třídu *StoreDelegate úložišti*.

Posluchači událostí jsou všechny komponenty, které implementují rozhraní *EventListener*. Potomci rozhraní *EventListener* jsou klasifikováni podle typu podtřídy *Event*, které naslouchají. Typicky se posluchač události implementuje jako vnitřní třída *managera* nebo aplikace, ze které jsou poté vyvolány příslušné služby na základě přijaté události.

5.2 Instalace

ONOS kontrolér je stejně jako ODL a Floodlight založený na programovacím jazyce Java. K jeho spuštění je tedy zapotřebí mít nainstalovaný JVM, který obsahuje například balíček JDK nebo JRE. Pro správnou a garantovanou funkčnost by měla by měla být nastavena proměnná prostředí JAVA_HOME, jedná se však pouze o doporučení.

Hardwarové požadavky závisí na mnoha faktorech, například na velikosti clusteru, velikosti spravované sítě, počtu zpráv vyměňovaných mezi zařízeními atd. Kontrolér má, stejně jako kontrolér ODL, vysoké nároky na operační paměť. Ideální velikost operační paměti je 4 GiB. Pokud je pro virtuální stroj vyhrazeno méně paměti, tak dochází ke zpomalení systému a získání kompletního pohledu na topologii trvá nepřípustně dlouho. Časové prodlevy jsou stejné jako u kontroléru ODL. Vzhledem k tomu, že je kontrolér ONOS napsán v jazyce Java, tak může být nainstalován a spuštěn na libovolném operačním systému. Oficiální dokumentace však doporučuje využít nejnovější LTS distribuci operačního systému Linux. Níže je uveden návod k instalaci ONOS kontroléru na distribuci Ubuntu (16.04).

Příkaz pro vytvoření speciálního neprivilegovaného uživatele. Vyžadováno scripty používanými ke spuštění ONOS kontroléru jako služby. Často vyžadován i specifický název účtu (sdn):

```
#sudo adduser sdn --system –group
```

Příkaz pro vytvoření adresáře `/opt`. Uživatelem vytvořený adresář se musí, na rozdíl od kontrolérů ODL, Floodlight a Ryu, nacházet v kořenovém adresáři systému:

```
#sudo mkdir opt
```

Příkaz pro stažení specifické verze balíčku ONOS (použita verze 1.8.2) ve formátu tar do adresáře `/opt`:

```
#sudo wget -c http://downloads.onosproject.org/release/onos-
$VERZE.tar.gz
```

Příkaz pro rozbalení staženého balíčku ONOS do adresáře `/opt`:

```
#sudo tar xzf onos-$VERZE.tar.gz
```

Příkaz pro přejmenovaní rozbaleného adresáře `onos-$VERZE` na `onos`:

```
#sudo mv onos-$VERZE onos
```

Po provedení všech výše uvedených kroků byl nainstalován základní ONOS kontrolér v takovém stavu, že je možné spustit jeho CLI.

5.3 Spouštění

Kontrolér ONOS podporuje na rozdíl od kontroléru ODL deaktivaci a kompletní odstranění aplikace z adresáře kontroléru. Aplikace kontroléru ONOS mohou být pouze nainstalovány nebo nainstalovány a aktivovány. Rozdíl mezi nainstalovanými a aktivovanými aplikacemi spočívá v tom, že nainstalované aplikace se pouze nachází v adresáři kontroléru a jsou připraveny k aktivaci. Aktivované aplikace spouští kontrolér při každém svém startu. Kontrolér ONOS po čisté instalaci obsahuje již několik nainstalovaných aplikací, nicméně tyto aplikace nejsou až na jednu výjimku, `org.onosproject.drivers` obsahující výchozí ovladače zařízení, aktivovány. Tudíž nebude fungovat ani spojení mezi OpenFlow zařízeními a kontrolérem ONOS. ONOS kontrolér jako jediný z porovnávaných kontrolérů vyžaduje při svém spuštění funkční připojení k internetu. Bez funkčního připojení nedojde při startu kontroléru k načtení aplikací a modulů a kontrolér se stává nepoužitelným. Po načtení veškerých modulů, aplikací a ověření funkčnosti je možné počítač od internetu odpojit.

ONOS kontrolér umožňuje aktivaci, deaktivaci a odstranění aplikací dvěma způsoby. Příslušnými příkazy v CLI kontroléru a skrze webové grafické rozhraní v záložce „Applications“. Instalaci lze realizovat pomocí webového grafického rozhraní přímo za běhu kontroléru v záložce „Applications“. Uživatel vybere možnost „upload an application (.oar)“, která je reprezentována tlačítkem „plus“. Následně vybere ve svém počítači požadovanou aplikaci, která je zabalena ve formátu oar (ONOS Application aRchive). Instalaci lze také realizovat pomocí manuálního nakopírování adresáře, který obsahuje aplikaci zabalenou ve formátu oar a je pojmenován `org.onosproject.$NAZEV_APLIKACE` do adresáře

`/opt/onos/apps/`. Při tomto postupu je však nutné kontrolér restartovat. Následující příkazy jsou zadávány v CLI kontroléru ONOS. Jejich kompletní výpis je možné nalézt na oficiálních stránkách kontroléru ONOS.[15]

Příkaz pro aktivaci aplikace:

```
#app activate org.onosproject.$NAZEV_APLIKACE
```

Příkaz pro deaktivaci aplikace:

```
#app deactivate org.onosproject.$NAZEV_APLIKACE
```

Příkaz pro kompletní odstranění aplikace:

```
#app uninstall org.onosproject.$NAZEV_APLIKACE
```

Příkaz pro výpis nainstalovaných aplikací (aktivované aplikace jsou označeny hvězdičkou):

```
#apps -s
```

Příkaz pro zobrazení aktivovaných aplikací:

```
#apps -s -a
```

ONOS kontrolér bylo navzdory oficiální dokumentaci nutné spouštět jako uživatel root. Při spuštění pod klasickým uživatelem kontrolér ONOS nenaslouchal ani na jednom z výchozích portů (6653, 6633) pro připojení k OpenFlow zařízením a virtualizovaná zařízení se tak nemohla spojit s kontrolérem. Příkaz pro spuštění kontroléru za předpokladu, že se uživatel nachází v jeho adresáři:

```
#sudo bin/onos-service start
```

5.4 Vlastnosti

Termín aplikace a modul se používá v dokumentaci ONOS kontroléru zaměnitelně. Stejně jako u ODL kontroléru budou aplikace a moduly představené v této podkapitole (tabulka 5) souhrnně nazývány vlastnostmi. Veškeré vlastnosti dodávané společně s ONOS kontrolérem je možné spravovat pomocí webového uživatelského rozhraní v záložce „Applications“ nebo skrze Karaf CLI ONOS kontroléru.

[15] https://wiki.onosproject.org/display/ONOS/Appendix+A+%3A+CLI+commands

Tabulka 5 – Vlastnosti dodávané společně s ONOS kontrolérem

Vlastnost	Název***
Access Control List*	org.onosproject.acl
Castor	org.onosproject.castor
Cluster IP alias	org.onosproject.cip
Control Plane Manager	org.onosproject.cpman
Distributed Load Test	org.onosproject.loadtest
Distributed Primitives Test	org.onosproject.distributedprimitives
DHCP Relay Agent	org.onosproject.dhcprelay
DHCP Server#	org.onosproject.dhcp
Driver Support Matrix	org.onosproject.drivermatrix
Event History	org.onosproject.events
Fault Management	org.onosproject.faultmanagement
Flow Performance Test	org.onosproject.flow-perf
Flow Space Analysis	org.onosproject.flowanalyzer
Flow Throughput Demo	org.onosproject.demo
Ganglia Report and Query	org.onosproject.gangliametrics
Graphite Report and Query	org.onosproject.graphitemetrics
Host Mobility	org.onosproject.mobility
InfluxDB Report and Query	org.onosproject.influxdbmetrics
Intent Performance Test App	org.onosproject.intentperf
Link Discovery Provider	org.onosproject.linkdiscovery
Master Election Test	org.onosproject.election
Mastership Load Balancer	org.onosproject.mlb
Messaging Performance Test	odl-tsdr-hsqldb-all
Multicast Forwarding	org.onosproject.mfwd
Network Config Host Provider	org.onosproject.netcfghostprovider
Network Config Link Provider	org.onosproject.netcfglinksprovider
Network Configuration Monitor Test	org.onosproject.netcfg-monitor
Null Southbound Provider	org.onosproject.null
OpenStack**	org.onosproject.openstackinterface, org.onosproject.metrics, org.onosproject.openstacknetworking , org.onosproject.openstacknode, org.onosproject.openstackrouting, org.onosproject.openstackswitching
Packet/Optical	org.onosproject.optical-model, org.onosproject.newoptical
Path Visualization	org.onosproject.pathpainter
Protocol Independent Multicast Emulation	org.onosproject.pim
Proxy ARP/NDP	org.onosproject.proxyarp
Reactive Forwarding#	org.onosproject.fwd
SDN-IP	org.onosproject.sdnip
SDN-IP Reactive Routing	org.onosproject.reactive-routing
Scalable Gateway	org.onosproject.scalablegateway
Segment Routing	org.onosproject.segmentrouting
Service function chaining**	org.onosproject.vtn
VLAN L2 Broadcast Network	org.onosproject.vpls

Vlastnost	Název***
Virtual Broadband Gateway	org.onosproject.virtualbng
Virtual Router	org.onosproject.vrouter
XOS Client	org.onosproject.xosclient
YANG Management System	org.onosproject.yms

*popsáno u Ryu kontroléru

#popsáno u Floodlight kontroléru

**popsáno u ODL kontroléru

***název/názvy sloužící k doinstalování dané vlastnosti skrze aplikační kontejner Karaf (některé vlastnosti se skládají z více částí)

Castor umožňuje *peering* mezi doménami (autonomními systémy), společnostmi a poskytovateli cloudových služeb. *Peering* znamená přímé spojení dvou či více síti bez nutnosti platit za přenos dat přes veřejný internet třetí straně. Poskytovatel/společnost získává díky tomu větší kontrolu nad přenosem dat. Nemůže tedy nastat situace, kdy jsou data ze dvou autonomních systému, které se nachází například v Praze, přenášena přes Londýn (Netnod, c2017).

Control Plane Manager slouží k monitorování řídící vrstvy a systému na kterém běží ONOS kontrolér. Umožňuje sbírat informace o počtu různých typů řídích zpráv přijatých/odeslaných kontrolérem (např. *packet-in, flow-mod*) a o využití systémových zdrojů. *Control Plane Manager* poskytuje REST API sloužící uživateli k dotazování a sbírání informací a je také integrován do webového uživatelského rozhraní. Počty různých zpráv jsou zde interpretovány jako sloupcové grafy. Informace jsou aktualizovány periodicky každou minutu.

DHCP Relay Agent slouží k přeposílání DHCP požadavků na DHCP server, který se nachází v jiné síti než host, který poslal požadavek. *DHCP Relay Agent* přijme *broadcast* DHCP požadavek od koncového zařízení a pošle ho jako *unicast* přímo DHCP serveru (Internet Research Task Force, 2001).

Event History slouží ke zobrazení historie událostí kontroléru ONOS.

Fault Management slouží k upozornění uživatele na vzniklé chyby na jednotlivých zařízeních. Zařízení při výskytu chyby pošle kontroléru upozornění, které je následně uloženo. *Fault Management* lze ovládat pomocí příkazového řádku ONOS kontroléru, webového uživatelského rozhraní skrze záložku „Alarms“ a REST API. Uživatel pomocí této vlastnosti může zjistit na kterém přepínači a portu k chybě došlo, přesný čas vzniku, čas vyřešení problému a typ chyby. Integrace do webového uživatelského rozhraní umožňuje zobrazit veškerá uložená upozornění v přehledné tabulce.

Ganglia Report and Query umožňuje periodicky posílat veškeré informace sesbírané kontrolérem ONOS (například konfiguraci sítě) monitorovacímu serveru Ganglia. Uložené informace je možné zobrazit ve webovém grafickém rozhraní Ganglia serveru. Ganglia je rozšiřitelný distribuovaný monitorovací systém určený pro vysoce výkonné výpočetní systémy. Před instalací samotné vlastnosti *Ganglia Report and Query* je zapotřebí mít nainstalovaný monitorovací server Ganglia.

Host Mobility slouží k automatickému mazání všech flow záznamů spojených s koncovým zařízením, které bylo odstraněno či přesunuto na jiné místo v síti.

InfluxDB Report and Query poskytuje stejné funkce jako *Ganglia Report and Query*. Jediný rozdíl je v tom, že data jsou periodicky posílány a ukládána v databázi InfluxDB. Uživatel může spárovat databázi InfluxDB s vizualizačním nástrojem třetí strany a poté si data zobrazit přehledně v grafickém rozhraní. Stejně jako u výše zmíněné *Ganglia Report and Query* je zapotřebí mít nainstalovanou a nakonfigurovanou InfluxDB nebo mít k této databázi vzdálený přístup.

Mastership Load Balancer umožňuje rovnoměrné vyvážení počtu zařízení, pro které bude kontrolér v rámci clusteru tzv. *master*. Role *master* znamená, že zařízení bude upřednostňovat právě tento kontrolér.

Multicast Forwarding slouží k reaktivnímu směrování *multicast* komunikace.

Network Config Host Provider umožňuje získávat informace o koncových zařízeních bez *packet-in* událostí. Umožňuje pomocí REST API nastavit koncová zařízení, která budou mít přístup do sítě. Ostatní koncová zařízení budou blokována.

Network Config Link Provider umožňuje pomocí REST API specifikovat spojení (určena pomocí identifikátorů koncových bodů) skrze která bude povolena komunikace. Nespecifikovaná spojení nebudou do topologie zahrnuta. Skrze nespecifikována spojení nemůže být vedena komunikace. Uživatel tak získává úplnou kontrolu nad topologií sítě.

Packet/Optical slouží ke zjednodušení správy vícevrstvé sítě. Jmenovitě sítě, která se skládá ze zařízení pro přenos pomocí optických kabelů a zařízení pro přenos pomocí elektrického signálu. Umožňuje inovativní konvergovaný pohled na topologii a rychlejší reakci na změny v síti. Je integrována do webového grafického rozhraní a umožňuje zobrazení obou vrstev sítě, vytváření spojení mezi jednotlivými koncovými body a upravování šířky pásma.

Path Visualization nebo také **Path Painter** umožňuje ve webovém grafickém rozhraní barevně odlišit různé typy cest mezi dvěma koncovými zařízeními. Uživatel může pomocí *Path Visualization* rozlišit tři typy cest – nejkratší, nesouvislou a nejkratší z geografického pohledu. *Path Visualization* je integrován do vizualizace topologie ve formě *overlay*.

Protocol Independent Multicast slouží k efektivnímu směrování *multicast* komunikace.

Proxy ARP/NDP umožňuje komunikaci koncových zařízení nacházejících se v jiných lokálních sítích tak, jako by se nacházela ve stejné síti. Pokud jsou dvě lokální sítě fyzicky připojené k routeru, na kterém je zapnuté *proxy ARP*, budou *broadcast* ARP (Address Resolution Protocol) dotazy posílány do sítě, ve které se koncové zařízení skutečně nachází. Zároveň bude router posílat svou MAC adresu jako odpověď. Tazatel poté bude komunikovat s routerem, který jeho dotazy přeposílá koncovému zařízení, se kterým chce komunikovat (Internet Research Task Force, 1987).

SDN-IP umožňuje připojit SDN síť k externím sítím pomocí BGP. Externím sítím spojeným s SDN sítí pomocí BGP protokolu se SDN síť jeví jako jediný autonomní systém. V rámci autonomního systému (SDN síť) poskytuje *SDN-IP* mechanismus, který umožňuje komunikaci mezi BGP a kontrolérem ONOS. Z perspektivy protokolu BGP se *SDN-IP* chová jako BGP *speaker*. Termínem BGP *speaker* se označuje router, na kterém běží BGP protokol. Z perspektivy kontroléru ONOS se jedná o vlastnost, která umožňuje vkládání a aktualizaci záznamů datové vrstvy (flow záznamy), které slouží ke směrování. *SDN-IP* může v rámci clusteru běžet ve více instancích, kde každá z těchto instancí obsahuje stejnou sadu BGP cest, avšak pouze jedna instance tzv. *SDN-IP Leader* vkládá a aktualizuje záznamy datové vrstvy. V případě výpadku hlavní instance je zvolen nový *SDN-IP Leader* a provedena synchronizace, která ověří, zdali záznamy datové vrstvy korespondují s aktuálními BGP cestami. *SDN-IP* se stará pouze o proaktivní instalaci cest pro tranzitní komunikaci. SDN síť v případě tranzitní komunikace slouží jako prostředník mezi dvěma autonomními systémy, jejichž koncová zařízení spolu komunikují. SDN síť v tomto případě není zdrojem ani cílem komunikace.

SDN-IP Reactive Routing slouží k reaktivním výpočtům a instalaci cest pro směrování IPv4 a IPv6 paketů. Plní též funkci virtuální výchozí brány pro koncová zařízení. Vlastnost je závislá na funkci *SDN-IP* a používá se ke směrování v případě, že zdrojem nebo cílem komunikace je koncové zařízení, které se nachází uvnitř SDN sítě, o zbytek se stará *SDN-IP*.

Scalable Gateway umožňuje nasazení více výchozích bran do SDN sítě a tím zajistit vyvažování zátěže, odolnost vůči chybám a vysokou dostupnost.

Segment Routing je zcela nový typ směrování, který lze přímo aplikovat na MPLS (Multiprotocol Label Switching) architekturu a jeho nasazení je výhodné zejména v SDN sítích. Paket je směrován jednotlivými uzly v síti pomocí seřazeného seznamu instrukcí, kterým se říká segmenty. Segmenty jsou umístěny v zásobníku a mohou obsahovat libovolnou instrukci (např. pošli paket specifickým rozhraním). Instrukce jsou prováděny v opačném pořadí, než byly do zásobníku vloženy. Poslední vložená instrukce se provádí jako první. Jakmile je instrukce provedena, tak je odstraněna ze zásobníku (Internet Research Task Force, 2017).

VLAN L2 Broadcast Network umožňuje uživateli vytvořit na OpenFlow infrastruktuře L2 *broadcast* doménu a přiřadit do ní určitá zařízení. Koncová zařízení spolu poté mohou komunikovat jako by byla součástí jedné *broadcast* domény. Jednotlivé domény se mohou vzájemně překrývat.

Virtual Broadband Gateway umožňuje připojení do internetu koncovým zařízením s privátní adresou. Přiděluje koncovým zařízením veřejné adresy z nakonfigurovaného rozsahu a uchovává mapování jednotlivých privátních adres na veřejné. Veškerá konfigurace je prováděna skrze poskytované REST API.

Virtual Router umožňuje nakonfigurovat OpenFlow přepínač tak, aby pracoval jako router.

YANG Management System umožňuje správu vlastností založených na modelovacím jazyce YANG.

5.5 Protokoly

V této podkapitole budou popsány protokoly (tabulka 6), které jsou dodávány společně s ONOS kontrolérem. Stejně jako vlastnosti, je i protokoly možné doinstalovat zvlášť skrze aplikační kontejner Karaf ve formě *southbound* zásuvných modulů.

Tabulka 6 – Protokoly dodávané společně s ONOS kontrolérem

Protokol	Název**
Border Gateway Protocol*	org.onosproject.bgp, org.onosproject.bgprouter
Intermediate System to Intermediate System Protocol	org.onosproject.isis
Network Configuration Protocol*	org.onosproject.netconf
Open vSwitch Database*	org.onosproject.ovsdb-base, org.onosproject.ovsdb, org.onosproject.ovsdbhostprovider
Path Computation Element Protocol	org.onosproject.pcep-api, org.onosproject.pcep
Simple Management Network Protocol*	org.onosproject.snmp
The Locator/ID Separation Protocol*	org.onosproject.lisp

*popsáno u ODL kontroléru

**název/názvy sloužící k doinstalování daného protokolu skrze aplikační kontejner Karaf (některé protokoly se skládají z více částí)

ISIS (Intermediate System to Intermediate System) je směrovací protokol sloužící k výměně směrovacích informací mezi směrovači v rámci autonomního systému. Jedná se o protokol stavu linky. Směrovače si vyměňují informace s nejbližšími sousedy. Topologickou informací je tímto způsobem zaplaven celý autonomní systém. Takže každý směrovač má kompletní pohled na celou topologii. ONOS kontrolér využívá tohoto protokolu k získání informací o topologii (Internet Research Task Force, 1990).

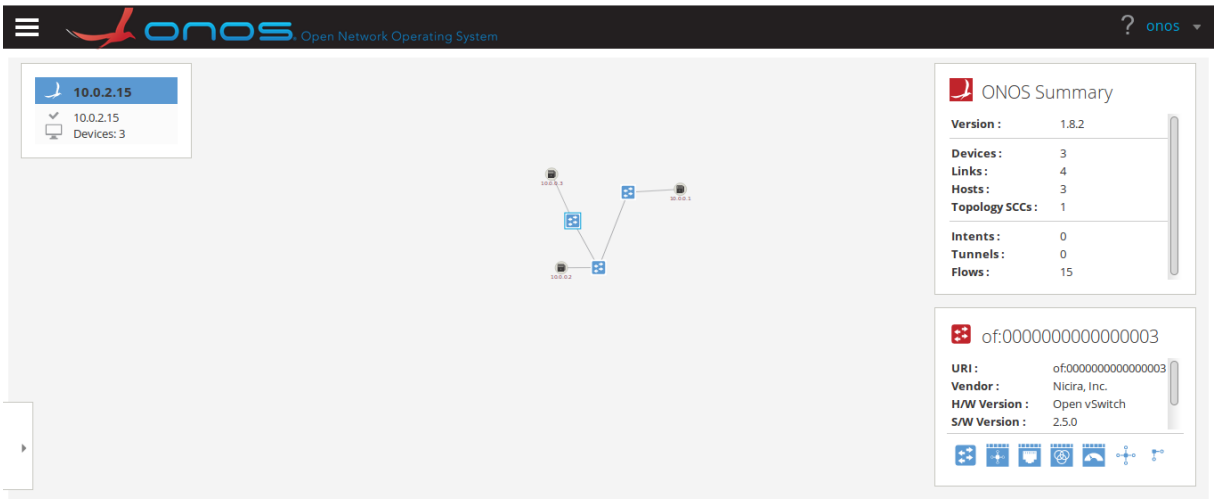
PCEP (Path Computation Element Protocol) slouží ke komunikaci mezi PCC (Path Computation Client) a PCE (Path Computation Element). PCE je síťovou komponentou, aplikací nebo uzlem, který dokáže vypočítat cestu od zdroje k cíli na základě topologie sítě a výpočetních omezení. PCC je libovolná aplikace požadující po PCE výpočet cesty. Tento model (PCC – PCE pomocí PCEP) se používá v MPLS a GMPLS (Generalized MPLS) sítích pro výpočet jednotlivých LSP (Label Switched Paths). Rozdíl mezi klasickým směrováním a MPLS spočívá v tom, že u MPLS jsou vytvořené cesty pro pár zdroj – cíl (LSP). Router díky tomu nemusí prohledávat směrovací tabulky. Místo směrovací tabulky využívá ke směrování záznamy o cestách, které jsou vedeny skrze něj (Internet Research Task Force, 2009b).

5.6 Grafické rozhraní

K uživatelskému rozhraní (obrázek 14) se přistupuje pomocí webového prohlížeče skrze URL *http://<ip adresa kontroléru>:8181/onos/ui/index.html*. Přístup do uživatelského rozhraní je zabezpečen pomocí uživatelského jména a hesla. Ve výchozím nastavení je uživatelského jméno „onos“ a heslo „rocks“. Grafické rozhraní kontroléru ONOS poskytuje znatelně

vylepšené verze funkcionalit popsaných u kontrolérů Floodlight a ODL. Dále přidává i funkce, které se u žádného kontroléru nevyskytují. Největší rozdíl je patrný ve vizualizaci topologie, která není pouze prostým zobrazením, ale poskytuje uživateli mnoho dalších funkcí. Umožňuje například zobrazení geografické mapy a možnost přemístit všechna zařízení na jejich skutečnou pozici, sledování objemu přenášených dat na uživatelem označených spojeních, rozlišení různých druhů spojení pomocí aplikace *Path Visualization*, skrývání/zobrazování informací (například unikátních identifikátorů přepínačů) a zařízení pomocí klávesových zkratek a zobrazení/skrývání různých vrstev (například optická vrstva) sítě. Nejdůležitější funkcí, kterou disponuje rozhraní ONOS kontroléru oproti ostatním je kompletní správa veškerých aplikací. Správa aplikací se nachází pod záložkou „Applications“. Ta umožňuje bez restartu kontroléru instalaci libovolné aplikace ve formátu oar. Dále řazení aplikací dle kategorií, vydavatele či podle abecedy, aktivaci, deaktivaci a kompletní odstranění vybrané aplikace.

Uživatelské jméno a heslo je možné změnit a na rozdíl od kontroléru ODL je ho možné i přidat. Pro správu uživatelů se u kontroléru ONOS využívá konfigurační soubor `/opt/onos/apache-karaf-3.0.5/etc/users.properties`. Všichni uživatelé se nachází na konci tohoto souboru. Uživatelská jména a hesla jsou nicméně uložena v plain textu. Takže si je každý, kdo má přístup k tomuto souboru, může přečíst a použít. Uživateli je možné nastavit jméno, heslo a skupinu. Na rozdíl od kontroléru ODL je po přidání, změně nebo smazání uživatele nutné restartovat kontrolér.



Obrázek 14 – Ukázka vizualizace topologie v grafickém rozhraní kontroléru ONOS

6 VÝVOJ APLIKACE PRO RŮZNÉ KONTROLÉRY

Cílem praktické části práce bylo vyvinout aplikaci, která dokáže získat souhrnné informace o topologii spravované libovolným analyzovaným kontrolérem. Získané informace jsou prezentovány pomocí grafického rozhraní, které též umožňuje uživateli s těmito informacemi dále pracovat. Uživatel se tak může například dotazovat kontroléru na detailní informace o flow záznamech na konkrétním zařízení. Pro práci s aplikací není vyžadována znalost způsobu komunikace s kontrolérem ani formátu unikátních identifikátorů jednotlivých zařízení u daného typu kontroléru.

Pro vývoj aplikace bylo použito vysokoúrovňového objektově orientovaného programovacího jazyka C#. Aplikace k získávání dat využívá *northbound* REST API, která jsou poskytována moduly kontrolérů. Pro vlastní přenos dat se používá HTTP (Hyper Text Transfer Protocol) protokolu. Aplikace posílá HTTP GET požadavek, na který dostává odpověď. Odpovědí jsou požadovaná data reprezentována formátem JSON. K testování funkcionalit aplikace byl použit síťový emulátor Mininet. Mininet umožňuje simulovat libovolnou topologii a umožnit její řízení libovolnému kontroléru. Pro účely testování však byly použity topologie již předpřipravené, které jsou dodávány společně s Mininetem ve formě Python skriptů.

6.1 Representational State Transfer API

REST je architektura rozhraní pro programování aplikací navržená pro distribuované prostředí. Poprvé představena v roce 2000 v disertační práci Roye Thomase Fieldinga. Umožňuje jednotný a snadný přístup ke zdrojům a provádět nad nimi CRUD operace. Jedná se o zkratku pro vytvoření (*create*), čtení (*read*), aktualizaci (*update*) a smazání (*delete*). Operace mohou být implementovány libovolnou metodou přenosu. Nejčastěji se však k jejich implementaci používá HTTP protokol (Hanák, 2013).

Zdrojem mohou být data, ale také stavy aplikace, pokud je lze popsat konkrétními daty. Každý zdroj má svůj vlastní identifikátor. REST nespecifikuje konkrétní syntaxi pro identifikaci zdrojů. Nejčastěji se však používá syntaxe URI (Uniform Resource Identifier). Jedná se o sekvenci znaků identifikující zdroj s pevně stanovenou syntaxí. Klient nepracuje přímo se zdroji, ale s jejich reprezentací. Nejčastěji používanými formáty pro reprezentaci zdrojů jsou JSON a XML (Hanák, 2013).

REST API je bezestavové. To znamená, že každý požadavek od klienta musí obsahovat veškeré informace k jeho pochopení serverem. Pokud server vyžaduje ověření klienta, je

zapotřebí ověřovat každý požadavek zvlášť. K ověření identity tedy nelze použít například relací či cookies (Hanák, 2013).

Pro účely testování komunikace s aplikacemi pomocí REST API je možné využít konzolového nástroje cURL. Jedná se o nástroj sloužící k přenosu dat ze serveru a na server pomocí jednoho z podporovaných protokolů. cURL podporuje například protokoly HTTP, IMAP, HTTPS a další (Stenberg, 2017).

Instalace cURL se provádí na linuxové distribuci Ubuntu pomocí následujícího příkazu:

```
#sudo apt install curl
```

6.2 Mininet

Mininet je síťový emulátor, který simuluje kolekci koncových zařízení, přepínačů a spojení mezi nimi na jednom linuxovém jádře. Používá lehkou virtualizaci k tomu, aby jeden systém vypadal jako kompletní síť běžící na stejném jádře, systému a uživatelském zdrojovém kódu. Běžně se používá k simulaci, verifikaci a testování. Jedná se o open source projekt hostovaný cloudovým repositářem Github (Nadeau & Gray, c2013, s. 86).

Hosté na Mininetu se chovají jako reálná zařízení, na kterých běží stejný kód. Každý host reprezentuje příkazový procesor virtuálního stroje. Mohou na něm běžet libovolné programy. Programy mohou posílat, přijímat a zpracovávat pakety v prostředí, které se jeví jako skutečný Ethernet, ale ve skutečnosti se jedná o virtuální zařízení nebo rozhraní. Pakety jsou zpracovávány virtuálními zařízeními, které se hostům jeví jako skutečné ethernetové přepínače nebo routery (Nadeau & Gray, c2013, s. 86).

6.2.1 Instalace

Podkapitola, pokud není uvedeno jinak, čerpá z oficiálních stránek síťového emulátoru Mininet (Mininet Team, c2017a).

V této podkapitole bude uveden podrobný návod, jak nainstalovat síťový emulátor Mininet na distribuci Ubuntu. Na oficiálních stránkách[16] lze také stáhnout předpřipravený obraz s předinstalovaným síťovým emulátorem Mininet. Tento obraz stačí pouze importovat do uživatelem zvoleného virtualizačního systému.

[16] https://github.com/mininet/mininet/wiki/Mininet-VM-Images

Příkaz pro instalaci programu git:

```
#sudo apt install git
```

Příkaz pro získání zdrojového kódu Mininetu:

```
#git clone git://github.com/mininet/mininet
```

Příkaz pro přesunutí do složky se staženým zdrojovým kódem:

```
#cd mininet
```

Příkaz pro zobrazení dostupných verzí:

```
#git tag
```

Příkaz pro výběr požadované verze:

```
#git checkout -b 2.2.2b2
```

Příkaz pro samotnou instalaci:

```
#mininet/util/install.sh [-parametry]
```

Následující parametry slouží ke specifikaci instalovaných položek:

- `-a` slouží k instalaci veškerých součástí a
- `-nfv` slouží k instalaci veškerých součástí, kromě POX kontroléru.

6.2.2 Spouštění

Podkapitola, pokud není uvedeno jinak, čerpá z oficiálních stránek síťového emulátoru Mininet (Mininet Team, c2017b).

V této podkapitole bude uveden postup, jak simulovat některou z předpřipravených topologií a umožnit její řízení uživatelem vybraným kontrolérem. Kontrolér může běžet i na jiném zařízení, než na kterém běží simulovaná topologie. Je však nutné zajistit mezi oběma zařízeními konektivitu.

Následující příkaz slouží k simulaci výchozí topologie o dvou koncových zařízeních a jednom přepínači. Mininet je dodáván společně s kontrolérem, který je ve výchozím nastavení použit k řízení topologie. Pokud chce uživatel využít k řízení topologie vlastního kontroléru, musí použít parametr `controller` a nastavit jeho hodnotu na `remote`. V případě, že vybraný kontrolér běží na stejném zařízení, je možné vynechat jeho IP adresu. Port je možné vynechat

v případě, že kontrolér naslouchá OpenFlow komunikaci na výchozích portech (6653 nebo 6633).

```
#sudo mn --controller=remote,ip=[IP adresa kontroléru],port=[číslo portu]
```

Přidání parametru `topo` umožňuje uživateli vybrat vlastní vytvořenou topologii nebo jednu z předpřipravených. Mininet umožňuje výběr z následujících tří druhů předpřipravených topologií:

- Minimální topologie. Používá se automaticky, když uživatel vynechá parametr *topo*. Lze ji nastavit pomocí hodnoty. Skládá se ze dvou hostů a jednoho přepínače.
- Lineární topologie. Mezi jednotlivými zařízeními existuje vždy pouze jedno spojení. Umožňuje specifikovat celkový počet zařízení.
- Stromová topologie. Vytváří logický strom. Umožňuje specifikovat počet pater stromu. Počet hostů je roven dvěma na počet pater. Počet přepínačů je roven dvěma na počet pater mínus jedna.

Parametr `topo` a jeho hodnoty (x je počet zařízení nebo pater):

```
--topo [minimal/linear,x/tree,x]
```

Před změnou topologie je zapotřebí použít následujícího příkazu. Pokud tento příkaz nebude zadán, tak bude kontrolér pravděpodobně zobrazovat jiný počet koncových zařízení a přepínačů, než se ve skutečnosti v simulované topologii nachází. Přeposílání a zpracování paketů bude navzdory tomu problému fungovat normálně.

```
#sudo mn –clean
```

Zbylé užitečné příkazy tykající se síťového emulátoru Mininet je možné nalézt na oficiálních stránkách.[17]

[17] http://mininet.org/walkthrough/

6.3 JSON

Podkapitola, pokud není uvedeno jinak, čerpá z webové stránky pojednávající o JSON formátu (Ecma International, 2013).

JSON je odlehčený, textový formát sloužící pro výměnu dat. Je založen na podmnožině programovacího jazyka JavaScript, Standard ECMA-262 3rd Edition. Nicméně se jedná o zcela jazykově nezávislý formát, který využívá programátorům velice dobře známé konvence z rodiny jazyků C. Pro člověka je snadno pochopitelný, čitelný a zapisovatelný. Strojově je ho možné dobře analyzovat i generovat.

6.3.1 JSON Text

JSON text je sekvence symbolů vytvořená z kódových bodů standardu Unicode, která odpovídá pevně dané syntaxi zápisu. Každý kódový bod odkazuje na určitý znak. Sada symbolů obsahuje šest symbolů definujících strukturu, řetězce, čísla a tři symboly pro doslovný zápis hodnoty (tzv. literály). Mezi symboly definující strukturu je možné vkládat bílé znaky.

6.3.2 JSON formát

JSON je založen na dvou následujících strukturách:

- Kolekci párů název/hodnota. Tato kolekce může být v různých programovacích jazycích realizována jako záznam, struktura, objekt, slovník, hash tabulka, asociativní pole nebo klíčový seznam.
- Seřazený seznam hodnot. Většina programovacích jazyků tento seznam realizuje jako seznam, vektor, posloupnost nebo pole.

Tyto datové struktury jsou zcela univerzální a podporované naprostou většinou moderních programovacích jazyků. V JSON jsou tyto struktury realizovány pomocí objektů a polí. Pole a objekty je možné do sebe vnořovat.

Objekt je reprezentován dvojicí složených závorek. Uvnitř složených závorek se nachází nula či více párů název/hodnota. Název je řetězec uzavřený do uvozovek. Každý název je následován dvojtečkou, za níž následuje hodnota. Jednotlivé páry název/hodnota jsou odděleny čárkami.

Pole je reprezentováno dvojicí hranatých závorek. Uvnitř hranatých závorek se nachází nula či více hodnot. Jednotlivé hodnoty jsou od sebe odděleny čárkami. Jedná se o seřazenou kolekci hodnot, důležité je tedy i jejich pořadí.

6.3.3 JSON hodnoty

JSON hodnotami mohou být čísla, řetězce, logické hodnoty (`true`, `false`), `null` nebo již výše zmíněné objekty a pole.

Čísla jsou zapisována v desítkové soustavě. Číslu může předcházet znaménko plus či mínus. Za číslem může být tečka, po níž následuje desetinná část čísla. Velká reálná čísla je možné zapisovat zkráceným zápisem pomocí mocnin deseti (e nebo E) následovaným znaménkem plus či mínus po kterém opět následuje číslo. Zápis v osmičkové a šestnáctkové soustavě není povolen.

Řetězce jsou sekvence znaků v kódování Unicode uzavřené ve dvojitých uvozovkách. Ve dvojitých uvozovkách mohou být umístěny všechny znaky. Pro umístění řídících znaků, dvojitých uvozovek a zpětného lomítka jsou používány tzv. únikové sekvence s použitím zpětného lomítka. To znamená, že každému takové znaku musí předcházet zpětné lomítko.

6.4 JSON.NET

Pro práci s daty ve formátu JSON je v aplikaci použito populárního frameworku Json.NET. Tento framework není implicitně dostupný a je ho zapotřebí nejprve stáhnout pomocí *NuGet Package Manager*. Teprve poté je možné využít metod a datových typů, které nabízí. Jedná se o volně dostupný framework, který vznikl v roce 2006. Json.NET nabízí následující funkce a výhody:

- Flexibilní JSON serializaci pro konverzi mezi .NET a JSON objekty.
- Umožňuje využití jazyka LINQ (Language Integrated Query) pro manuální čtení a zápis formátu JSON.
- Poskytuje rychlejší serializaci než vestavěný .NET sloužící ke stejnému účelu.
- Umožňuje konverzi z JSON do XML a obráceně.
- Podporuje širokou škálu verzí .NET, *Windows Phone* a *Windows 8 Store* (Newtonsoft, c2017a).

V aplikaci je k získávání hodnot z JSON textu použito výhradně jazyka LINQ, neboť pro získání souhrnných informací o topologii není zapotřebí serializovat celý JSON na .NET objekt. Z celého JSON textu je zapotřebí získat pouze několik konkrétních hodnot a provést tak manuální serializaci JSON na .NET objekt.

6.4.1 Manuální serializace JSON pomocí LINQ

LINQ je označení pro soubor nástrojů sloužící k dotazování na data. Jedná se o dotazovací jazyk integrovaný přímo do syntaxe jazyka C#. Umožňuje zjednodušení a zobecnění práce s libovolným typem dat. Jedná se o deklarativní jazyk, velice podobný jazyku SQL (Structured Query Language). Poskytuje jeden unifikovaný způsob dotazování na data. Pomocí jazyka LINQ je tedy možné například stejným způsobem hledat prvek v poli, číst data z XML souboru a hledat uživatele v databázi (Čápka, 2013).

Json.NET framework umožňuje použití jazyka LINQ pro dotazování se na data v JSON formátu. Poskytuje následující metody pro získávání hodnot z JSON objektů a polí:

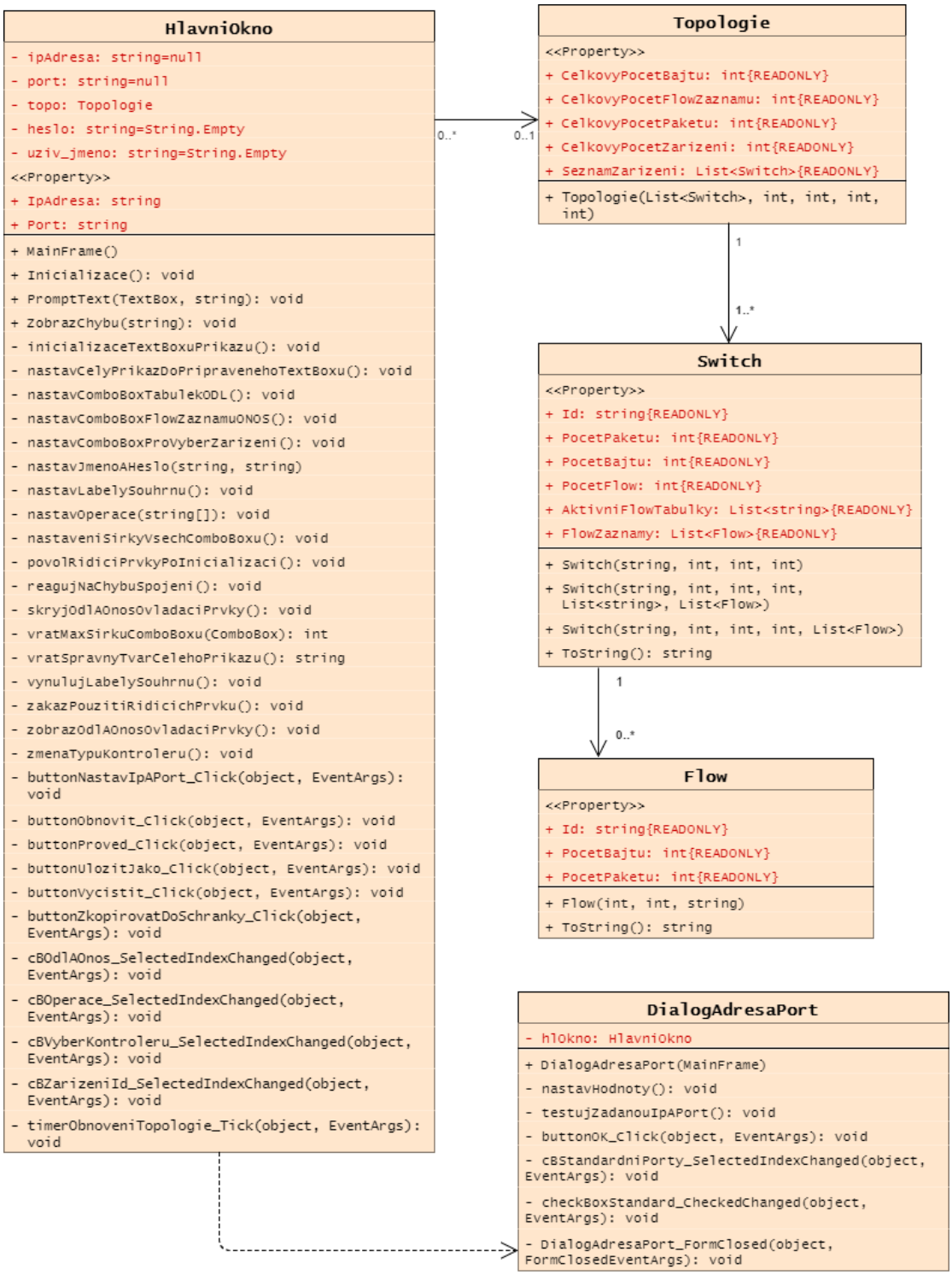
- Získávání hodnot z objektů a polí pomocí indexů. U objektů se do hranatých závorek uvádí jméno vlastnosti, jejíž hodnotu chce uživatel získat. Zatímco u polí se uvádí klasické kladné celé číslo v rozmezí nula až délka pole mínus jedna.
- Získávání hodnot z objektů a polí pomocí metody `Children`. U objektu metoda vrací kolekci jeho vlastností a u pole vrací kolekci jeho hodnot. S těmito kolekcemi je poté možné dále pracovat jako s libovolným druhem kolekce v jazyce C#. Je možné ji například procházet s využitím `foreach` (Newtonsoft, c2017b).

Json.NET poskytuje několik nových datových typů, jež jsou v rámci aplikace použity. Jmenovitě se jedná o tyto datové typy:

- `JArray`,
- `JObject`,
- `JValue` a
- `JProperty`.

6.5 Popis jednotlivých tříd aplikace

Aplikaci tvoří celkem devět tříd. Přičemž instanční třídy ukazuje UML (Unified Modeling Language) diagram, který je zobrazen na obrázku 15. Zbylé třídy jsou statické a nelze tedy od nich vytvořit instanci.



Obrázek 15 – UML diagram tříd

6.5.1 Formulářové třídy

Aplikace obsahuje formulářové třídy `HlavniOkno` a `DialogAdresaPort`, sloužící uživateli k ovládání aplikace. Obsahují ovládací prvky s registrovanými událostmi a implementaci jejich obslužných rutin.

`HlavniOkno` zastává funkci hlavního formuláře aplikace, který se objeví okamžitě po jejím spuštění. Slouží k vytvoření a udržování instance třídy `Topologie`, jejíž atributy jsou zobrazovány uživateli ve formě souhrnných topologických informací. Umožňuje uživateli posílat specifické webové požadavky kontroléru a jejich zobrazení v čitelné podobě.

`DialogAdresaPort` slouží jako dialogové okno pro zadání IP adresy a portu kontroléru. Kromě obslužných rutin je zde implementována metoda pro kompletní otestování vstupů od uživatele. Uživateli tak není dovoleno zadat například adresu ve špatném formátu či kombinaci portu a adresy, na které se nenachází žádný kontrolér.

6.5.2 Třídy reprezentující objekty počítačové sítě

K reprezentaci reálných objektů počítačové sítě slouží instance tříd `Topologie`, `Switch` a `Flow`. Tyto třídy mají implementovány vlastnosti, které dané objekty počítačové sítě popisují. Vlastnosti jsou získávány manuální serializací JSON formátu.

Třída `Switch` slouží k reprezentaci konkrétního přepínače, který se nachází v topologii spravované kontrolérem. Instance třídy udržuje následující informace o konkrétním přepínači:

- unikátní identifikátor přepínače v rámci topologie,
- počet paketů a bajtů zpracovaný přepínačem,
- počet flow záznamů uložených ve všech flow tabulkách přepínače,
- seznam unikátních identifikátorů aktivních flow tabulek přepínače (platí pouze pro kontrolér OpenDaylight) a
- seznam veškerých flow záznamů uložených přepínačem (platí pouze pro kontroléry OpenDaylight a ONOS).

Třída `Flow` slouží k reprezentaci konkrétního flow záznamu uloženého ve flow tabulce přepínače. Instance třídy udržuje o konkrétním flow záznamu následující informace:

- unikátní identifikátor flow záznamu v rámci určitého přepínače,
- počet paketů a bajtů zpracovaných pomocí flow záznamu.

Třída `Topologie` slouží k reprezentaci logické topologie spravované libovolným kontrolérem. Instance třídy udržuje o logické topologie následující informace:

- celkový počet bajtů a paketů zpracovaný všemi přepínači v topologii,
- celkový počet flow záznamů na všech přepínačích v topologii,
- celkový počet přepínačů nacházejících se v topologii a
- seznam veškerých zařízení v topologii.

6.5.3 Statické třídy

Statické třídy nacházející se v aplikaci slouží ke zpřehlednění zdrojového kódu. Obsahují pouze pomocné metody a atributy.

Třída `NacitaniJSON` obsahuje metodu pro získání dat z určité URL (Uniform Resource Locator) adresy ve formátu JSON a metodu pro úpravu libovolného JSON textu do čitelného formátu, který je poté prezentován uživateli v rámci hlavního formuláře.

Třídy `ONOS`, `Ryu`, `Floodlight` a `Opendaylight` obsahují pomocné atributy a metodu `VytvorTopologii` týkající se konkrétního typu kontroléru. Metoda `VytvorTopologii` slouží k manuální serializaci formátu JSON a vrací vytvořenou instanci třídy `Topologie`. Každá z těchto tříd obsahuje následující atributy:

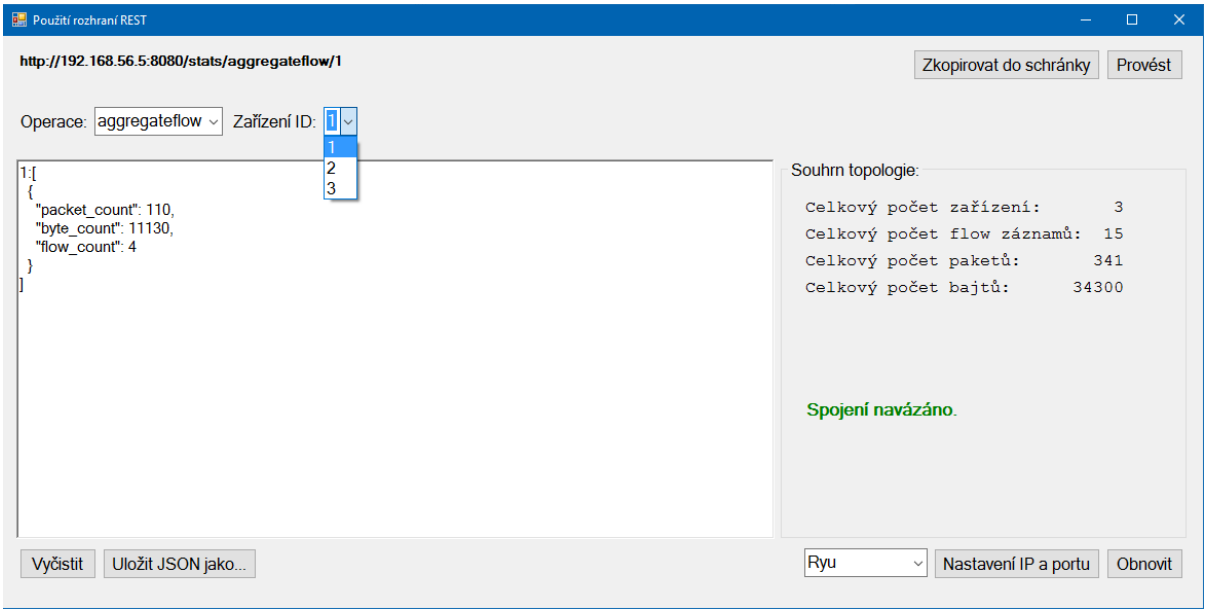
- neměnnou část URL adresy specifickou pro daný typ kontroléru a
- pole řetězců obsahující operace, které daný typ kontroléru umožňuje.

6.6 Grafické rozhraní aplikace

Grafické rozhraní aplikace se skládá z jednoho hlavního okna a dialogu pro zadávání IP adresy a portu kontroléru. K tvorbě grafického rozhraní byl použit .NET framework Windows Forms. Jednotlivé ovládací prvky hlavního okna a dialogu se zpřístupňují a zakazují dle akcí vykonaných uživatelem tak, aby bylo okamžitě vidět, které z nich má smysl v dané chvíli použít. Veškeré chybové stavy aplikace jsou uživateli prezentovány ve formě modálních oken.

6.6.1 Hlavní formulář

Design hlavního formuláře je responzivní a umožňuje tak uživateli libovolně zvětšit či zmenšit okno, aniž by docházelo k deformaci či ztrátě ovládacích prvků. Vzhled a dostupné funkce hlavního formuláře se dynamicky mění podle vybraného typu kontroléru, při vzniku chyby a podle akcí provedených uživatelem. Například šířka rozbalovacích seznamů se dynamicky mění podle nejdelšího řetězce, který rozbalovací seznam obsahuje. Okno hlavního formuláře se uživateli objeví okamžitě po startu aplikace. Téměř veškeré ovládací prvky však budou zakázány. Povolen bude pouze rozbalovací seznam sloužící k výběru typu kontroléru a tlačítko „Nastavení IP a portu“. Uživatel tedy nejprve musí nastavit požadovaný typ kontroléru a pomocí dialogu nastavit IP adresu a port. Na obrázku 16 je znázorněn formulář s již vytvořenou instancí třídy `Topologie`.



Obrázek 16 – Hlavní formulář

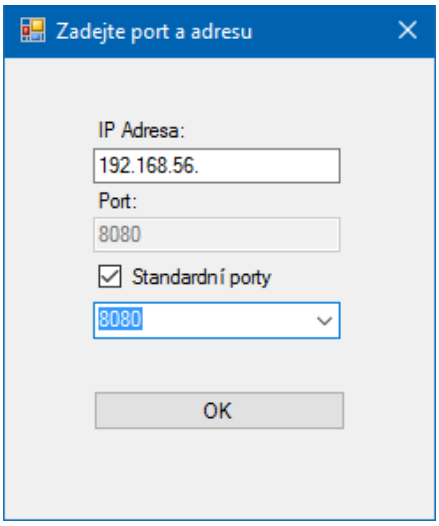
Hlavní formulář po zadání správné IP adresy, portu a typu kontroléru zobrazuje souhrnné informace o spravované topologii. Pro automatickou aktualizaci informací o topologii slouží časovač. Informace jsou aktualizovány každých 30 vteřin.

Po vytvoření instance třídy `Topologie` umožňuje hlavní formulář uživateli například:

- Posílat specifické webové požadavky kontroléru a odpovědi následně zobrazovat v čitelné podobě v textovém poli.
- Ukládání obsahu textového pole do souboru.
- Zkopírování URL adresy do schrány pro provedení dotazu pomocí webového prohlížeče.

6.6.2 Dialog pro zadání IP adresy a portu

Dialog pro zadání IP adresy a portu (obrázek 17) musí uživatel použít při každé změně kontroléru a při spuštění aplikace.



Obrázek 17 – Dialog sloužící k zadávání adresy a portu

Z důvodu zdlouhavého zadávání IP adresy je zde automaticky doplněna její část, kterou má ve výchozím nastavení síťový adaptér pro komunikaci hostitelského systému s virtualizovaným. Rozbalovací seznam umožňuje volit mezi standardními porty, které kontroléry používají ve výchozím nastavení. Po odškrtnutí zaškrtávacího pole „Standardní porty“ je uživateli umožněno zadat vlastní port z rozsahu 0 až 65535. Uživateli není dovoleno zadat port nebo IP adresu ve špatném formátu. Pokud uživatel zadá neplatnou IP adresu nebo port, tak je na to upozorněn skrze modální okno.

6.7 Získání surového JSON

Aplikace je z pohledu architektury REST API klientem, který se dotazuje kontroléru pomocí HTTP požadavků na určitá data. Kontrolér odpovídá na tyto požadavky a zasílá zpět aplikaci odpověď ve formátu surového nezpracovaného řetězce. Tento řetězec reprezentuje JSON text. Pro posílání webových požadavků a získání nezpracovaného JSON textu slouží statická metoda `Nacti`, která je umístěna ve statické třídě `NacitaniJSON`. Tato metoda je využívána k získání nezpracovaných dat napříč všemi druhy kontrolérů.

Metoda `Nacti` má tři vstupní parametry. Parametr `adresa` slouží ke specifikaci URL adresy, na které se nachází požadovaná data. Tato adresa se u jednotlivých kontrolérů a požadovaných dat liší. Dále zde jsou parametry `uziv_jmeno` a `heslo`, které slouží k autorizaci webového požadavku. Jednotlivé požadavky je zapotřebí autorizovat pouze

u kontrolérů ONOS a OpenDaylight. Načítání musí být obaleno blokem `try-catch`, pro případ náhlého výpadku spojení mezi kontrolérem a aplikací.

```
public static string Nacti(string adresa,string uziv_jmeno, string heslo)
            //vytvoření webového požadavku
            WebRequest pozadavek = WebRequest.Create(adresa);
            //autorizace příkazů REST
            //pokud není zadáno heslo a jméno, tak je autorizace vynechána
            if (uziv_jmeno.Length>0 && heslo.Length > 0)
            {
                //řetězec pro ověření identity
                string overeni_identity = uziv_jmeno + ":" + heslo;
                CredentialCache pamet = new CredentialCache();
                //zakódování řetězce pro ověření pomocí Base64
                pozadavek.Headers["Authorization"] = "Basic " +
Convert.ToBase64String(Encoding.ASCII.GetBytes(overeni_identity));
            }
            //získání odpovědi
            WebResponse odpoved = pozadavek.GetResponse();
            //získání proudu dat
            Stream proudDat = odpoved.GetResponseStream();
            //přečtení proudu dat
            StreamReader reader = new StreamReader(proudDat,
            Encoding.UTF8);
            string surovyJson = reader.ReadToEnd();
            return surovyJson;
```

6.8 Parsování JSON

Parsování formátu JSON je v aplikaci provedeno pomocí výše zmíněného volně dostupného frameworku Json.NET. V podstatě vše co aplikace umožňuje souvisí s parsováním formátu JSON. Ať už se jedná o souhrnné informace o topologii, které zobrazuje grafické rozhraní nebo naplnění rozbalovacích seznamů například kolekcemi unikátních identifikátorů zařízení nebo flow záznamů. V aplikaci se nachází celkem pět statických metod, které provádějí parsování formátu JSON.

6.8.1 Metoda pro úpravu JSON textu

Poslední metodou, která se týká parsování formátu JSON, je metoda `vratNaparsovanyJson`, která se nachází ve statické třídě `NacitaniJSON`. Tato metoda slouží k úpravě libovolného JSON textu do dobře čitelného a upravitelného formátu. Přidává do JSON řetězce znaky, které slouží k odřádkování jednotlivých párů název/hodnota. Takto upravený řetězec je poté metodou vrácen a je připraven ke zobrazení v textovém poli hlavního formuláře.

Metoda `vratNaparsovanyJson` požaduje pouze jeden vstupní parametr a tím je neupravený JSON text. Nejprve je zapotřebí zjistit, zdali se jedná o pole, pokud ne, tak je na něj JSON text manuálně převeden přidáním hranatých závorek. Poté se použije metoda `Parse` třídy

`JArray`, která převede JSON text na `JArray`. Všechny objekty nacházející se v `JArray` je možné projít jako libovolnou kolekci pomocí `foreach`. Následně se provede průchod veškerými vlastnostmi, které objekt obsahuje a ty jsou postupně přidávány do připraveného prázdného řetězce. Nakonec je vrácen řetězec, kde se na každém řádků nachází jeden pár název/hodnota.

Takto upravený řetězec je daleko lépe čitelný a upravitelný, neboť v nezpracovaném JSON textu se vše nachází na jednom řádků.

```
public static string vratNaparsovanyJson(string jsonN)
            string naparsovanyJson = "";
            string nenaparsovanyJson = jsonN;
            /*pokud JSON nemá spravný formát (pole), je třeba doplnit na
            */začátek [ a  na konec ]
            if (jsonN[0] != '['){
               nenaparsovanyJson = "[" + nenaparsovanyJson + "]";
            }
            //naparsování JSON textu na JArray
            //přidává odřádkování
            JArray jsonObjekty = JArray.Parse(nenaparsovanyJson);
            //pruchod vsemi Json objekty --uvozeny { a } ukončeny
            foreach (JObject objekt in jsonObjekty.Children<JObject>())
            {
                foreach (JProperty vlastnost in objekt.Properties())
                {
                    //pruchod vsemi vlastnostmi - jmeno:hodnota
                    string jmeno = vlastnost.Name;
                    string hodnota = vlastnost.Value.ToString();
                    /*pridani vlastnosti a hodnot ve spravnem formatu do
                    retezce*/
                    naparsovanyJson += jmeno + ":" + hodnota;
                }
            }
            return naparsovanyJson;
        }
```

6.8.2 Vytvoření instance třídy Topologie

Metoda `VytvorTopologii` slouží k získání požadovaných souhrnných informací o topologii. Pomocí těchto informací je následně vytvořena instance třídy `Topologie`, která je datovou složkou hlavního formuláře. Její atributy jsou poté zobrazeny uživateli v rámci topologického souhrnu, který se nachází v pravé části hlavního formuláře. Zároveň také slouží k naplnění veškerých rozbalovacích seznamů sloužících uživateli ke specifikaci webového požadavku na určité informace o topologii. Například pro zobrazení detailních informací o určitém flow záznamu na konkrétním zařízení.

Metoda `VytvorTopologii` má čtyři vstupní parametry:

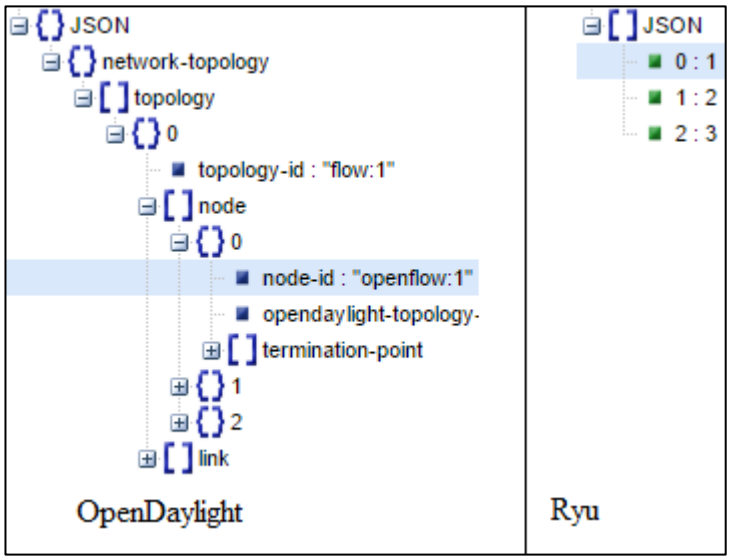
- `ip` sloužící ke specifikaci IP adresy kontroléru,
- `port` sloužící ke specifikaci portu kontroléru a
- `uziv_jmeno` a `heslo` sloužící k autorizaci webového požadavku.

Níže uvedený zdrojový kód je pouze úryvkem sloužícím k demonstraci manuální serializace JSON formátu na .NET objekt. Úryvek zobrazuje způsob získání unikátních identifikátorů přepínačů, které se nacházejí v topologii spravované kontrolérem OpenDaylight. Pracuje se zde s datovými typy `JObject` a `JArray`. Tyto datové typy v kombinaci s integrovaným jazykem LINQ umožňují jednotný přístup k požadovaným datům. Manuální serializace a získání pouze určitých hodnot vyžaduje znalost struktury konkrétního JSON textu. Je nutné znát název a umístění požadovaného páru název/hodnota. Zdali se požadované páry nachází v poli či objektu.

```
public static Topologie VytvorTopologii(string ip, string port,string
                                        uziv_jmeno,string heslo)
            //seznam pro ukládání dpid zařízení
            List<string> id_zarizeni = new List<string>();
            //získání formátu JSON s informacemi o topologii
            string json = NacitaniJSON.Nacti("http://" +ip + ":" + port
            +"/restconf/operational/network-topology:network-
            topology",uziv_jmeno,heslo);
            //převedení JSON na JObject
            JObject j_objekt_topologie = JObject.Parse(json);
            //získání JArray s topologiemi
            JArray poleTopologii = (JArray)j_objekt_topologie["network-
            topology"]["topology"];
            //průchod veškerými topologiemi spravovanými kontrolérem
            foreach (JObject topo in poleTopologii.Children<JObject>())
            {
                //získání hodnoty vlastnosti node
                JArray poleUzlu = (JArray)topo["node"];
                //průchod všemi uzly v topologii
                foreach (JObject uzly in poleUzlu.Children<JObject>())
                {
                    //získáni hodnoty vlastnosti dpid
                    string dpid = (string)uzly["node-id"];
                    //filtrování síťových a koncových zařízení
                    //dpid síťového zařízení začíná podřetězcem openflow
                         if (dpid.Substring(0, 8) == "openflow"){
                            id_zarizeni.Add(dpid);
                         }
                }
            }
```

6.8.3 Různá struktura JSON formátu

Odpověď na webový požadavek na stejný typ informací je každým z analyzovaných kontrolérů zasílána ve formě různě strukturovaného JSON souboru. Hodnoty stejných vlastností se nachází pod jinými názvy nebo jsou vnořeny do dalších struktur. Tento problém je možné ilustrovat na příkladu dvou JSON souborů, které obsahují unikátní identifikátory veškerých zařízení, která se nachází v topologii.



Obrázek 18 – Formát JSON kontrolérů ODL a Ryu

Strukturu JSON souborů kontrolérů OpenDaylight a Ryu znázorňuje obrázek 18. Oba tyto soubory poskytují stejné informace. Rozdíl je v tom, že OpenDaylight navíc k požadovaným unikátním identifikátorům poskytuje ještě další informace. Samotné unikátní identifikátory se nacházejí pod názvem „node-id“ a jsou zanořeny v šesti strukturách. Naproti tomu Ryu poskytuje pouze pole, ve kterém se na každém indexu nachází jeden unikátní identifikátor. Díky absenci jakékoli standardizace bylo nutné implementovat čtyři odlišné varianty metody `VytvorTopologii`. Pro každý kontrolér jednu. Z toho vyplývá, že přidání dalšího typu kontroléru by vyžadovalo naimplementovat novou variantu metody `VytvorTopologii`.

7 ZÁVĚR

V teoretické části práce byla provedena analýza jednotlivých kontrolérů a kapitoly jsou strukturovány takovým způsobem, aby si čtenář mohl danou část či postup jednoduše porovnat s dalším analyzovaným kontrolérem. Bohužel to dokumentace jednotlivých kontrolérů ne vždy umožňovaly. Analýza se nicméně neopírá pouze o teoretické znalosti citované z dokumentací jednotlivých kontrolérů. Každý z analyzovaných kontroléru jsem nainstaloval a poté s ním pracoval, abych tak mohl čtenáři podat informace založené na vlastních zkušenostech. Největším problémem při analýze jednotlivých kontrolérů byl nedostatek informací o možných problémech a jejich řešeních. Z důvodu nedostatku informací bylo řešení každého problému časově velmi náročné.

Při analýze jsem zjistil, že kontroléry Floodlight a Ryu nevykazují, na rozdíl od kontrolérů ONOS a ODL, žádné problémy, mají nízké hardwarové nároky a přehledné dokumentace. Na druhou stranu však nenabízí zdaleka tolik funkcionalit. Kontroléry Floodlight a Ryu se používají pouze pro experimentální a studijní účely. Naproti tomu kontroléry ONOS a ODL jsou již nasazeny i v produkčních prostředích. Na ODL kontroléru je dokonce založen komerční kontrolér od společnosti Cisco. Je tedy možné, že se problémy týkají pouze konkrétních verzí kontrolérů nebo použitého síťového emulátoru Mininet.

Při vytváření aplikace jsem se naučil pracovat s rozhraním REST a formátem JSON. Největší překážkou při vývoji aplikace byla absence standardizace rozhraní REST. Každý kontrolér má data se stejným významem na jiných adresách a jsou také jinak strukturována. Pro každý kontrolér bylo tedy nutné implementovat vlastní metodu pro získávání informací o topologii. Využití aplikace je v současném stavu v praxi nepoužitelné. Aplikace byla vytvořena pouze za účelem porovnání funkcionalit jednotlivých kontrolérů. Pokud by se však aplikace zaměřila pouze na jeden kontrolér, tak by ji bylo možné použít v praxi, rozšířit ji o spoustu dalších funkcionalit a upravit grafické rozhraní tak, aby si uživatel neuvědomoval existenci rozhraní REST.

8 POUŽITÁ LITERATURA

Apache Ant – Welcome. *Welcome to The Apache Software Foundation!* [online]. The Apache Software Foundation, c2017 [cit. 2017-04-21]. Dostupné z: http://ant.apache.org/.

Architecture. *RYU SDN Framework* [online]. RYU project team, c2014b [cit. 2017-04-20]. Dostupné z: https://osrg.github.io/ryu-book/en/html/arch.html.

ČÁPKA, David. 7. díl – LINQ v C# – revoluce v dotazování. *Itnetwork.cz - Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další* [online]. itnetwork.cz, 2013 [cit. 2017-04-18]. ISSN 2464-6326. Dostupné z: http://www.itnetwork.cz/csharp/kolekce-a-linq/c-sharp-tutorial-linq-dotazy.

Difference between JDK, JRE and JVM – javatpoint. *Tutorials – Javatpoint* [online]. Javatpoint, c2011–2017 [cit. 2017-04-21]. Dostupné z: https://www.javatpoint.com/difference-between-jdk-jre-and-jvm.

Download/Get Started with Mininet – Mininet. *Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet* [online]. Mininet Team, c2017a [cit. 2017-04-18]. Dostupné z: http://mininet.org/download/.

Draft-ietf-spring-segment-routing-11 - Segment Routing Architecture. *IETF Tools* [online]. Internet Research Task Force, 2017 [cit. 2017-04-27]. Dostupné z: https://tools.ietf.org/html/draft-ietf-spring-segment-routing-11.

Extending Python with C or C++ – Python 2.7.13 documentation. *Welcome to Python.org* [online]. Python Software Foundation, c1990–2017 [cit. 2017-04-21]. Dostupné z: https://docs.python.org/2/extending/extending.html.

Floodlight OpenFlow Controller – Project Floodlight. *Home – Project Floodlight – OpenFlow news and projectsProject Floodlight | Open Source Software for Building Software-Defined Networks* [online]. Project Floodlight, c2017 [cit. 2017-04-21]. Dostupné z: http://www.projectfloodlight.org/floodlight/.

Getting Started – Ryu 4.13 documentation. *Welcome to RYU the Network Operating System(NOS) – Ryu 4.13 documentation* [online]. Nippon Telegraph and Telephone Corporation, c2011–2014a [cit. 2017-04-20]. Dostupné z: http://ryu.readthedocs.io/en/latest/getting_started.html.

GORANSSON, Paul a Chuck BLACK. *Software defined networks: a comprehensive approach*. Waltham: Morgan Kaufmann, c2014. ISBN 978-0-12-416675-2.

Guides – ONOS – Wiki. *Wiki Home – ONOS – Wiki* [online]. Onos Community, 2017 [cit. 2017-04-21]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Wiki+Home.

HANÁK, Drahomír. Stopařův průvodce REST API. *Itnetwork.cz - Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další* [online]. itnetwork.cz, 2013 [cit. 2017-04-18]. ISSN 2464-6326. Dostupné z: http://www.itnetwork.cz/nezarazene/stoparuv-pruvodce-rest-api.

Introduction. *Json.NET - Newtonsoft* [online]. Newtonsoft, c2017a [cit. 2017-04-18]. Dostupné z: http://www.newtonsoft.com/json/help/html/Introduction.htm.

JSON. *JSON* [online]. Ecma International, 2013 [cit. 2017-04-18]. Dostupné z: http://www.json.org/json-cz.html.

Mininet Walkthrough – Mininet. *Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet* [online]. Mininet Team, c2017b [cit. 2017-04-18]. Dostupné z: http://mininet.org/walkthrough/.

Message-Oriented Middleware (MOM) (Sun Java System Message Queue 4.3 Technical Overview). *Oracle | Integrated Cloud Applications and Platform Services* [online]. Oracle Corporation and/or its affiliates, 2010 [cit. 2017-04-23]. Dostupné z: https://docs.oracle.com/cd/E19340-01/820-6424/aeraq/index.html.

NADEAU, Thomas D. a Kenneth GRAY. *SDN: software defined networks*. Sebastopol: O´Reilly Media, c2013. ISBN 978-1-449-34230-2.

ONF Overview – Open Networking Foundation. *Home – Open Networking Foundation* [online]. Open Networking Foundation, c2017 [cit. 2017-03-22]. Dostupné z: https://www.opennetworking.org/about/onf-overview.

Platform Overview | OpenDaylight. *The OpenDaylight Platform | OpenDaylight* [online]. OpenDaylight Project, c2016 [cit. 2017-04-21]. Dostupné z: https://www.opendaylight.org/platform-overview/.

Project Floodlight. *Project Floodlight* [online]. Atlassian, c2017 [cit. 2017-04-21]. Dostupné z: https://floodlight.atlassian.net/wiki/spaces/HOME/overview.

Querying JSON with LINQ. *Json.NET - Newtonsoft* [online]. Newtonsoft, c2017b [cit. 2017-04-18]. Dostupné z: http://www.newtonsoft.com/json/help/html/QueryingLINQtoJSON.htm.

RFC 1027 - Using ARP to implement transparent subnet gateways. *IETF Tools* [online]. Internet Research Task Force, 1987 [cit. 2017-04-27]. Dostupné z: https://tools.ietf.org/html/rfc1027.

RFC 1142 - OSI IS-IS Intra-domain Routing Protocol. *IETF Tools* [online]. Internet Research Task Force, 1990 [cit. 2017-04-27]. Dostupné z: https://tools.ietf.org/html/rfc1142.

RFC 3046 - DHCP Relay Agent Information Option. *IETF Tools* [online]. Internet Research Task Force, 2001 [cit. 2017-04-27]. Dostupné z: https://tools.ietf.org/html/rfc3046.

RFC 5415 - Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification. *IETF Tools* [online]. Internet Research Task Force, 2009a [cit. 2017-04-23]. Dostupné z: https://tools.ietf.org/html/rfc5415.

RFC 5440 - Path Computation Element (PCE) Communication Protocol (PCEP). *IETF Tools* [online]. Internet Research Task Force, 2009b [cit. 2017-04-27]. Dostupné z: https://tools.ietf.org/html/rfc5440.

RFC 6241 - Network Configuration Protocol (NETCONF). *IETF Tools* [online]. Internet Research Task Force, 2011 [cit. 2017-04-23]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc6241.

RFC 6830 - The Locator/ID Separation Protocol (LISP). *IETF Tools* [online]. Internet Research Task Force, 2013 [cit. 2017-04-23]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc6830.

RFC 7285 - Application-Layer Traffic Optimization (ALTO) Protocol. *IETF Tools* [online]. Internet Research Task Force, 2014 [cit. 2017-04-23]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc7285.

RFC 7426 - Software-Defined Networking (SDN): Layers and Architecture Terminology. *IETF Tools* [online]. Internet Research Task Force, 2015a [cit. 2017-03-22]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc7426.

RFC 7665 - Service Function Chaining (SFC) Architecture. *IETF Tools* [online]. Internet Research Task Force, 2015b [cit. 2017-04-23]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc7665.

ROUSE, Margaret. What is machine-to-machine (M2M)? - Definition from WhatIs.com. *TechTarget*[online]. TechTarget, 2010 [cit. 2017-04-23]. Dostupné z: http://internetofthingsagenda.techtarget.com/definition/machine-to-machine-M2M.

ROUSE, Margaret. What is multi-tenancy? - Definition from WhatIs.com. *TechTarget* [online]. TechTarget, 2014 [cit. 2017-04-23]. Dostupné z: http://whatis.techtarget.com/definition/multi-tenancy.

ROUSE, Margaret. What is plane (in networking)? - Definition from WhatIs.com. *TechTarget* [online]. TechTarget, 2013 [cit. 2017-03-22]. Dostupné z: http://whatis.techtarget.com/definition/plane-in-networking.

Ryu.app.ofctl_rest – Ryu 4.13 documentation. *Welcome to RYU the Network Operating System(NOS) – Ryu 4.13 documentation* [online]. Nippon Telegraph and Telephone Corporation, c2011–2014b [cit. 2017-04-20]. Dostupné z: http://ryu.readthedocs.io/en/latest/app/ofctl_rest.html.

RYU Controller Tutorial | SDN Hub. *SDN Hub | Software-defined Networking forum* [online]. SDN Hub, c2014 [cit. 2017-04-20]. Dostupné z: http://sdnhub.org/tutorials/ryu/.

RYU project team. *RYU SDN Framework* [online]. 1. Ryu SDN Framework Community, c2014a [cit. 2017-04-20]. Dostupné z: https://osrg.github.io/ryu-book/en/Ryubook.pdf.

SDN Architecture Overview. In: *Home – Open Networking Foundation* [online]. Open Networking Foundation, c2013 [cit. 2017-03-22]. Dostupné z: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf.

SRIDHAR, Rao. SDN Series Part Five: Floodlight, an OpenFlow Controller – The New Stack. *Home – The New Stack* [online]. The New Stack, 2015a [cit. 2017-04-21]. Dostupné z: https://thenewstack.io/sdn-series-part-v-floodlight/.

SRIDHAR, Rao. SDN Series Part Seven: ONOS – The New Stack. *Home – The New Stack* [online]. The New Stack, 2015b [cit. 2017-04-21]. Dostupné z: https://thenewstack.io/sdn-series-part-v-floodlight/.

STENBERG, Daniel. Curl – How To Use. Curl [online]. 2017 [cit. 2017-04-18]. Dostupné z: https://curl.haxx.se/docs/manpage.html.

Understanding the SDN Architecture – Definition. *SDxCentral Trusted News and Resources for SDx, SDN, NFV* [online]. SDNCentral, c2012–2017a [cit. 2017-03-22]. Dostupné z: https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture.

What is OpenStack Neutron? *SDxCentral Trusted News and Resources for SDx, SDN, NFV* [online]. SDNCentral, c2012–2017b [cit. 2017-04-23]. Dostupné z: https://www.sdxcentral.com/cloud/open-source/definitions/what-is-openstack-quantum-neutron/.

What is Open vSwitch Database or OVSDB? - Definition. *SDxCentral Trusted News and Resources for SDx, SDN, NFV* [online]. SDNCentral, c2012–2017c [cit. 2017-04-23]. Dostupné z: https://www.sdxcentral.com/cloud/open-source/definitions/what-is-ovsdb/.

What is peering | Netnod. *Netnod* [online]. Netnod, c2017 [cit. 2017-04-27]. Dostupné z: https://www.netnod.se/ix/what-is-peering.