Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Vývoj SDN aplikací pro ONOS kontrolér

Bc. Jan Mokráček

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2018/2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Jan Mokráček**
Osobní číslo:	**I17215**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Vývoj SDN aplikací pro ONOS kontrolér**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem diplomové práce je popsat základní architekturu, vlastnosti a funkcionality SDN kontroléru ONOS. Popis bude obsahovat praktický postup zprovoznění kontroléru v emulovaném prostředí Mininet na vlastní síťové topologii. Praktická část práce se zaměří na vytvoření vlastního modulu pro tento kontrolér. Modul bude poskytovat monitorování datových toků v síťové topologii s možností ručního povolení či zakázání jednotlivých komunikací. Modul bude ovládán přes grafické rozhraní a monitorování toků bude navíc podporováno i v prostředí příkazového řádku.

Rozsah grafických prací:

Rozsah pracovní zprávy: **50**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

OPEN NETWORKING FOUNDATION OpenFlow Switch Specification: Version 1.5.1 (Protocol version 0x06)[online]. 2015 [cit. 2018-09-21]. Dostupné z: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
ONOS Developer Guide [online]. 2017 [cit. 2018-09-21]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Developer+Guide
NADEAU, Thomas D a Kenneth GRAY. SDN: software defined networks Sebastopol, CA: O'Reilly Media, 2013. ISBN 978-1-4493-4230-2
GORANSSON, Paul a Chuck BLACK Software defined networks: a comprehensive approach, Amsterdam: Morgan Kaufmann, c2014. ISBN 978-0-12-416675-2

Vedoucí diplomové práce: **Ing. Filip Holík, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **22. října 2018**
Termín odevzdání diplomové práce: **18. května 2019**

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 17. listopadu 2018

Prohlašuji:

Tuto práci jsem 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.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 8. 5. 2019

Jan Mokráček

PODĚKOVÁNÍ

Chtěl bych poděkovat Ing. Filipovi Holíkovi, Ph.D. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování diplomové práce věnoval. Děkuji také své rodině a přátelům za podporu při studiu.

ANOTACE

Teoretická část práce se zabývá obecným popisem architektury, hlavních subsystémů a funkcionalit SDN kontroléru ONOS. Součástí popisu je také praktický postup, jak ONOS kontrolér zprovoznit a jak s ním pracovat. V praktické části práce je nejprve uveden obecný postup, jak vyvíjet moduly pro ONOS kontrolér. Následně je popsán vývoj vlastního modulu Traffic Analyzer, který poskytuje monitorování datových toků v síťové topologii s možností ručního povolení a zakázání komunikace.

KLÍČOVÁ SLOVA

Monitoring datových toků, ONOS, SDN, vývoj modulů.

TITLE

Development of SDN applications for ONOS controller

ANNOTATION

The theoretical part deals with general description of architecture, main subsystems and functionalities of the ONOS SDN controller. The description also includes how to make the ONOS controller operational and how to work with it. The practical part firstly describes the development of modules for the ONOS controller in general. Subsequently, the development of the Traffic Analyzer module is described, which provides traffic flow monitoring in network topology with the possibility of manually enabling and disenabling communication.

KEYWORDS

Development of modules, ONOS, SDN, traffic flow monitoring.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

Artemis	Automatic and Real-Time detection and Mitigation System
AS	Autonomní systém
ARP	Address Resolution Protocol
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BDDP	Broadcast Domain Discovery Protocol
BGP	Border Gateway Protocol
CLI	Command Line Interface
CFM	Connectivity Fault Management
CORD	Central Office Re-architected as a Datacenter
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DOCSIS	Data Over Cable Service Interface Specification
DoS	Denial of Service
DPID	Datapath ID
DSL	Domain-specific language
eBGP	external BGP
FTP	File Transfer Protocol
FPGA	Field Programmable Gate Array
GPON	Gigabit Passive Optical Network
GRE	Generic Routing Encapsulation
GUI	grafické uživatelské rozhraní
HTTP	Hypertext Transfer Protocol
iBGP	internal BGP
IXP	Internet Exchange Point
ICMP	Internet Control Message Protocol

IP	Internet Protocol
IMR	Intent Monitor and Reroute service
JAR	Java Archive
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
L2	Layer 2
L3	Layer 3
L7	Layer 7
LAN	Local Area Network
LLDP	Link Layer Discovery Protocol
LTS	Long Term Support
MAC	Media Access Control
MAN	Metropolitan Area Network
ML2	Modular Layer 2
MML	Man Machine Language
MQ	Message Queue
NDP	Neighbor Discovery Protocol
NETCONF	Network Configuration Protocol
OAR	ONOS Application aRchive
ODTN	Open and Disaggregated Transport Network
ONF	Open Networking Foundation
ON.Lab	Open Networking Lab
ONOS	Open Source Network Operating System
OVSDB	Open vSwitch Database
P4	Programming Protocol-independent Packet Processors
PIM	Protocol-Independent Multicast
POM	Project Object Model

RAM	Random Access Memory
REST	Representational State Transfer
SDN	Softwarově definovaná síť
SMTP	Simple Mail Transfer Protocol
SONA	Simplified Overlay Network Architecture
SPA	Single page application
TAPI	Transport API
TCP	Transmission Control Protocol
TL1	Transaction Language 1
UDP	User Datagram Protocol
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VPN	Virtuální privátní síť
VxLAN	Virtual Extensible LAN
WAN	Wide Area Network
XML	Extensible Markup Language
XOS	Extensible Service Control Plane

ÚVOD

Množství přenášených dat v moderních datových sítích neustále roste, stejně jako očekávání jejích uživatelů na dostupnost, rychlost a nízkou latenci. Vzhledem k uvedeným neustále rostoucím požadavkům naráží klasická síťová architektura již na své limity. Jedno z možných řešení představují softwarově definované sítě (SDN), jejichž základním aspektem je oddělení datové vrstvy od řídící. Řídící vrstva je u SDN vyňata z hardwaru jednotlivých zařízení a je přesunuta do jednoho centralizovaného prvku, tzv. kontroléru.

Kontrolér provádí řízení sítě pomocí SDN aplikací, které mohou být interní či externí. Interní aplikace jsou spravovány a spouštěny samotným kontrolérem, zatímco externí aplikace komunikují s kontrolérem pomocí rozhraní, která umožňují komunikaci v síťovém prostředí. Výhodou interních aplikací je minimální latence, zatímco výhodou externích je možnost volby programovacího jazyka, ve kterém bude externí aplikace vyvíjena. Tato diplomová práce je zaměřena na vývoj interních SDN aplikací, které jsou v rámci této práce označovány jako moduly.

Toto téma bylo zvoleno z důvodu jeho aktuálnosti, kterou dokládají výsledky průzkumu provedeného společností IHS Markit v roce 2018 (Weissberger, 2019). V rámci tohoto průzkumu bylo dotázáno 23 poskytovatelů služeb z celého světa, kteří dohromady představují 44 % telekomunikačního trhu, jakým způsobem využívají nebo zdali mají v plánu využívat SDN. Přičemž všech 23 dotázaných poskytovatelů služeb uvedlo, že v blízké budoucnosti plánuje postupně migrovat od klasické síťové architektury k SDN a 78 % plánuje jejich nasazení již do konce roku 2018. Nasazení SDN jde však ruku v ruce s výběrem vhodného kontroléru a vývojem vlastních modulů.

Dostupných kontrolérů, pro které je možné vyvíjet vlastní moduly, je celá řada – např. Floodlight, Ryu, Open Source Network Operating System (ONOS), OpenDaylight, Cisco Open SDN Controller a další. Nicméně vzhledem k tomu, že SDN jsou využívány v současné době zejména v produkčních prostředích s velkým množstvím síťových prvků, byl pro vývoj vybrán ONOS kontrolér, který je určený primárně pro poskytovatele služeb a datová centra.

Cílem teoretické části práce je popis ONOS kontroléru. První dvě kapitoly jsou věnovány popisu jeho architektury. Přičemž první kapitola je věnována jejímu obecnému představení a druhá podrobnému popisu hlavních subsystémů, ze kterých se ONOS kontrolér skládá. Třetí kapitola je věnována praktickému postupu, jak zprovoznit ONOS kontrolér a jak s ním pracovat. Čtvrtá kapitola je věnována popisu dostupných funkcionalit, které jsou dodávány společně s ONOS kontrolérem ve formě tzv. ONOS aplikací (moduly představují jejich podmnožinu).

V praktické části práce je popsán vývoj vlastního modulu Traffic Analyzer, který poskytuje monitorování datových toků v síťové topologii s možností ručního povolení a zakázání komunikace. Před popisem vývoje vlastního modulu Traffic Analyzer je v páté kapitole uveden obecný postup, jak vyvíjet moduly pro ONOS kontrolér.

1 OPEN SOURCE NETWORK OPERATING SYSTEM

Architektura sítí poskytovatelů služeb musí být navržena tak, aby uspokojila exponenciálně rostoucí požadavky na šířku pásma. Ty jsou způsobeny zejména rostoucím počtem mobilních zařízení a distribucí obsahu v cloudu. Následkem zvyšujících se požadavků na šířku pásma a rostoucího počtu připojených zařízení je zvýšení zátěže síťových zařízení. Současná síťová prostředí závisí na třech vzájemně propojených atributech: škálovatelnosti, výkonu a vysoké dostupnosti (Subramanian a Voruganti, 2016, s. 151; Goransson, Black a Culver, c2017, s. 180).

ONOS je open source projekt organizace Open Networking Lab (ON.Lab), který byl zahájen v roce 2014. ON.Lab je nezisková organizace založena investory ochotnými investovat do SDN a Berkeleyskou a Stanfordovou univerzitou. Na vývoji ONOS kontroléru se podílejí poskytovatelé služeb (American Telephone and Telegraph, Nippon Telegraph and Telephone, Verizon), výrobci síťových zařízení (Ciena, Cisco, Ericsson, Fujitsu, Huawei, Intel, NEC, Nokia), provozovatelé sítí (Internet2, CNIT, CREATE-NET) a další. Projekt také podporuje nezisková organizace Open Networking Foundation (ONF), která se zaměřuje na rozvoj SDN a standardizaci protokolu OpenFlow[1] a souvisejících technologií. V roce 2015 se stal ONOS součástí Linux Foundation, která podporuje vytváření udržitelných otevřených ekosystémů poskytováním finančních a intelektuálních zdrojů, infrastruktury, školení atd. (Open Networking Foundation, c2019; About The Linux Foundation, c2019).

ONOS je od základu vytvořen jako kompletní síťový operační systém poskytující funkcionality, které klasické SDN kontroléry[2] neposkytují. Na obrázku 1 je znázorněna evoluce SDN kontroléru včetně funkcionalit, které poskytují jednotlivé evoluční stupně. ONOS zahrnuje veškeré funkcionality všech evolučních stupňů. Jedná se tedy o první operační systém SDN navržený pro extrémně spolehlivé (*carrie-grade*) a *mission critical*[3] sítě, který je určený zejména pro poskytovatele služeb. Je implementován v programovacím jazyce Java a spadá pod Apache licenci verze 2.0 (Rao, 2015; Subramanian a Voruganti, 2016, s. 151–152).

Z hlediska SDN kontrolérů je ONOS rozšiřitelný, modulární a distribuovaný kontrolér celé SDN sítě. Poskytuje abstraktní rozhraní pro programování SDN aplikací a rozhraní pro správu, monitoring a programování SDN zařízení. Podporuje konfiguraci sítě založenou na síťových politikách (tzv. *policy-based networking*[4]) a zpracování síťových událostí. Síťové politiky jsou vytvářeny pomocí aplikačních záměrů, které jsou vysvětleny v kapitole 2.4 (Coker a Azodomolky, 2017, s. 188; Subramanian a Voruganti, 2016, s. 152).

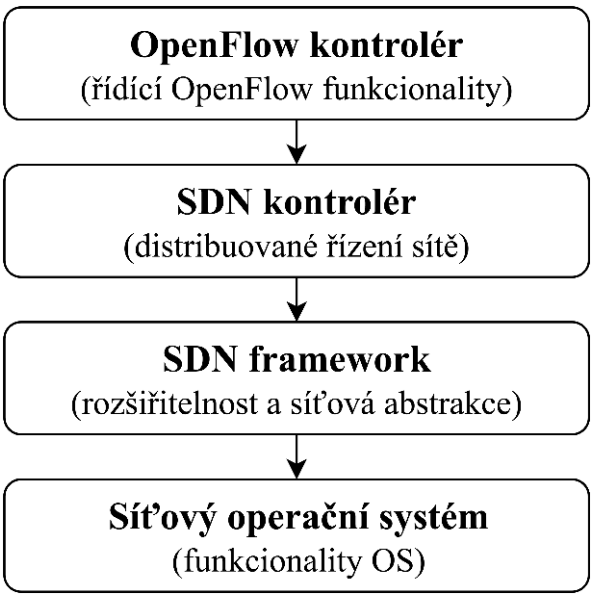
---

[1] OpenFlow je první neproprietární protokol sloužící k programování datové vrstvy SDN zařízení. Definuje jak komunikační protokol mezi řídící a datovou vrstvou, tak i část chování samotné datové vrstvy SDN architektury (Goransson, Black a Culver, c2017, s. 89–94).

[2] Kontrolér je software, který udržuje kompletní pohled na topologii, implementuje rozhodovací politiku sítě (např. směrování), řídí všechna SDN zařízení v topologii a poskytuje *northbound a southbound* Application Programming Interface (API). *Southbound* API umožňuje programování SDN zařízení a *northbound* API umožňuje začlenit aplikace do kontroléru (Goransson, Black a Culver, c2017, s. 64–77).

[3] *Mission critical* je taková síť, která musí fungovat nepřetržitě a spolehlivě. Výpadek může znamenat velké finanční ztráty, poškození reputace společnosti nebo dokonce ztráty na životech (Rouse, 2014a).

[4] *Policy-based networking* je takový typ správy sítě, kde různým druhům komunikace (například data, hlas, video) jsou přiřazeny takové priority a taková šířka pásma, aby síť fungovala z hlediska pohledu uživatele efektivně. Například, aby nedocházelo k výpadkům hlasové komunikace (Rouse, 2007).



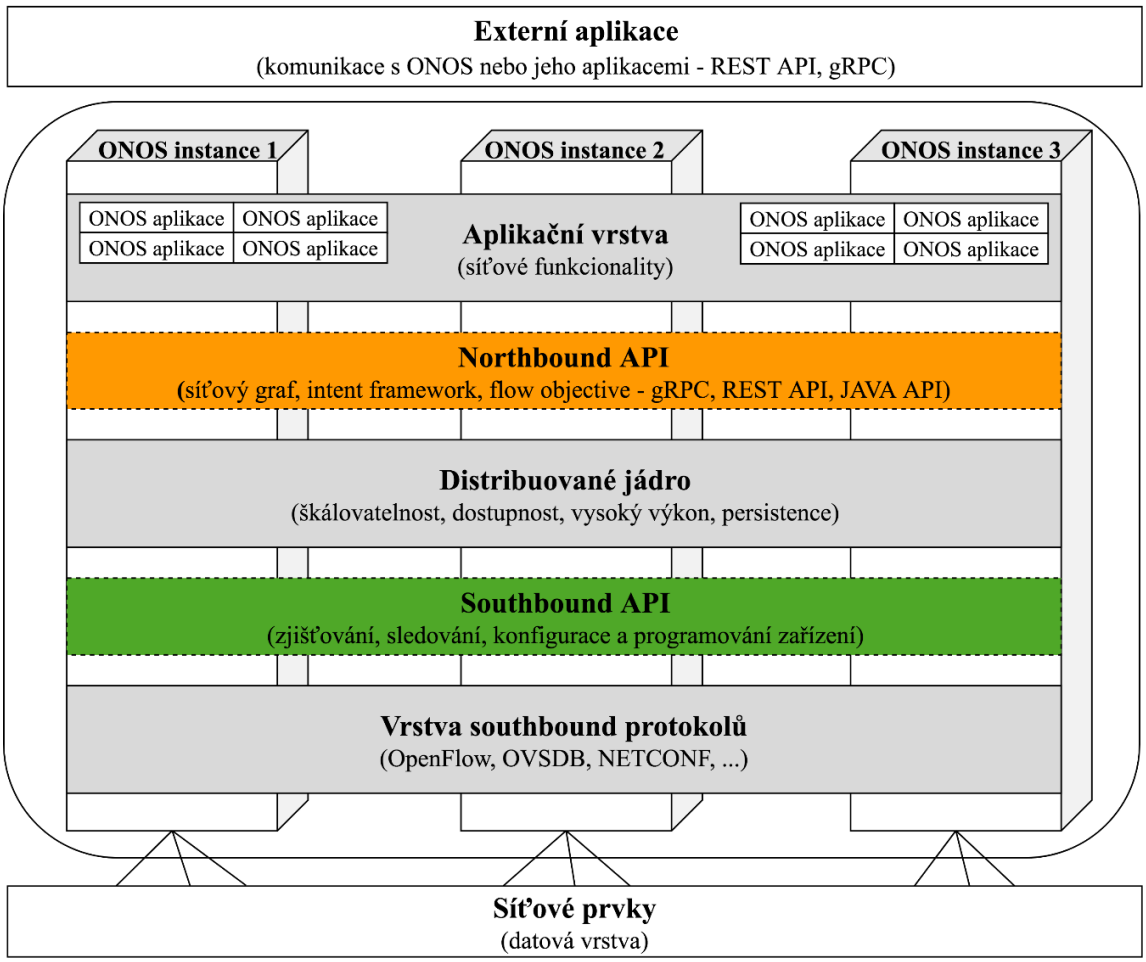
Obrázek 1: Evoluce SDN kontrolérů

Zdroj: zpracováno podle (Subramanian a Voruganti, 2016, s. 152)

Z hlediska serverových operačních systémů poskytuje ONOS analogické funkcionality. Mezi tyto funkcionality patří například: abstrakce a alokace zdrojů, izolace, virtualizace, řízení přístupu a software orientovaný na uživatele, jako je například Command Line Interface (CLI), grafické uživatelské rozhraní (GUI) a systémové aplikace. ONOS se na rozdíl od operačních systémů síťových zařízení tradiční síťové architektury (např. Cisco IOS) používá ke správě celé sítě a nikoli jednoho zařízení. Správa sítě jako celku umožňuje zjednodušení její správy a konfigurace a usnadňuje nasazení nového softwaru, hardwaru a služeb (Coker a Azodomolky, 2017, s. 188; Subramanian a Voruganti, 2016, s. 152).

1.1 Architektura

ONOS architektura je navržena speciálně pro požadavky extrémně spolehlivých a *mission critical* sítí (vysoký výkon, vysoká dostupnost a škálovatelnost). Architektura má velmi dobře definované abstrakce a skládá se ze tří vrstev, které jsou znázorněny na obrázku 2 šedou barvou. Interakce mezi jednotlivými vrstvami zprostředkovávají rozhraní znázorněná oranžovou a zelenou barvou. Na ONOS architekturu lze také nahlížet jako na třístupňovou kolekci subsystémů, která je znázorněna na obrázku 3 (Rao, 2015; Coker a Azodomolky, 2017, s. 188, Subramanian a Voruganti, 2016, s. 153).



Obrázek 2: Architektura ONOS kontroléru

Zdroj: vlastní zpracování

ONOS zahrnuje čtyři hlavní prvky, které utváří jeho architekturu. Jedná se o:

- **Distribuované jádro**, které je nezávislé na použitém *southbound* protokolu a slouží ke sledování a poskytování stavu sítě aplikacím skrze *northbound* API. Distribuované jádro je nejdůležitější architektonickou vlastností z pohledu poskytovatele služeb. Operační systém SDN je od počátku navržen pro běh v clusteru[5]. Díky clusteru je možné implementovat následující požadavky na extrémně spolehlivé sítě: šířka pásma, agilita, pružnost, odolnost vůči chybám, vysoký výkon a elastická škálovatelnost založená na aplikaci. Všechny instance ONOS kontroléru, které utváří cluster mají stejné informace o topologii a jsou na nich nasazeny stejné aplikace (organizace clusteru je podrobně popsána v kapitole 1.2). Tyto aplikace ani síťová zařízení nemají povědomí o tom, zda spolupracují s jednou či několika instancemi ONOS kontroléru. Díky tomu může síťový administrátor libovolně škálovat síť.

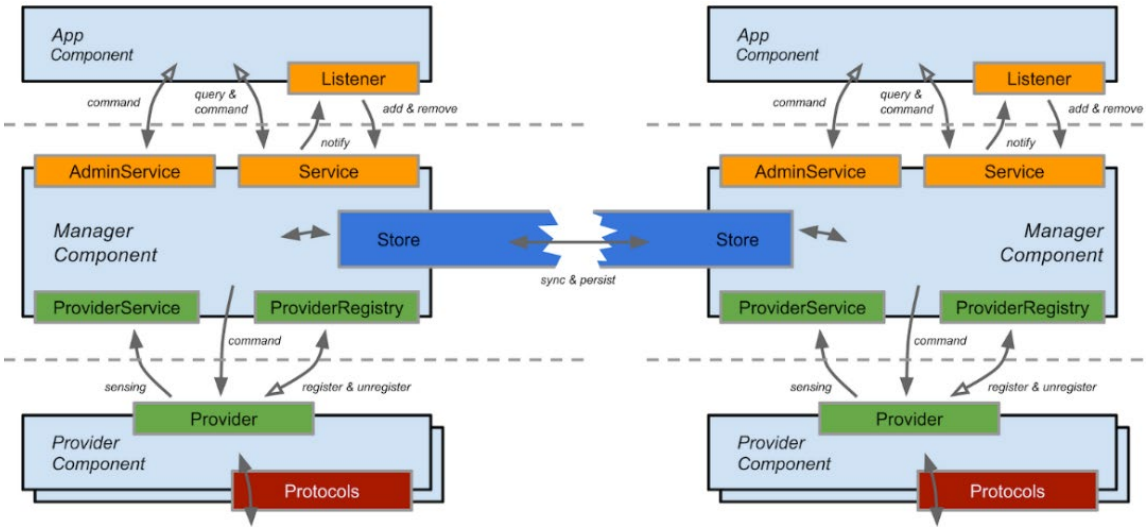
[5] Cluster je skupina serverů a dalších zdrojů, které se chovají jako jeden systém. Použitím clusteru je umožněno dosažení vysoké dostupnosti, vyvažování zátěže a paralelní zpracování (Rouse, 2006).

- ***Northbound* API** poskytuje konfigurační a řídící služby pro vývoj SDN aplikací. Poskytuje aplikacím tři druhy abstrakcí: *Intent framework*, který umožňuje aplikacím požadovat určité služby sítě bez předchozí znalosti, jak budou dané služby poskytovány. Správci sítí a vývojáři aplikací tuto výhodu využívají při vysokoúrovňovém síťovém programování. Stačí pouze specifikovat, co má síť provést (například vytvořit tunel mezi dvěma koncovými zařízeními) a vývojáře či administrátora už nemusí zajímat, jakým způsobem síť jeho požadavků dosáhne. *Globální pohled na síť*, který aplikacím poskytuje pohled na kompletní topologii sítě ve formě síťového grafu. *Flow objective*, který umožňuje vývojářům aplikací programování SDN zařízení nezávisle na typu SDN zařízení a použitém *southbound* protokolu.
- ***Southbound* API společně s vrstvou *southbound* protokolů** slouží k izolaci jádra ONOS kontroléru od detailů různých protokolů a zařízení tím, že každý síťový prvek reprezentují jako obecný objekt. Tato abstrakce umožňuje distribuovanému jádru udržovat stav síťového uzlu bez znalostí jeho specifických detailů. *Southbound* API umožňuje správu různých zařízení pomocí více různých protokolů bez zásahu do jádra a jednoduché přidávání nových zařízení. Podpora pro jednotlivé protokoly se díky modularitě architektury instaluje ve formě zásuvných modulů. Ve výchozím nastavení jsou podporovány protokoly: OpenFlow, Open vSwitch Database (OVSDB) a Network Configuration Protocol (NETCONF).
- **Modulární architektura softwaru** je navržena pro efektivní vývoj, nasazení a údržbu softwaru. Usnadňuje vývoj aplikací a služeb open source komunitě vytvořené kolem ONOS kontroléru (Coker a Azodomolky, 2017, s. 189–190, Subramanian a Voruganti, 2016, s. 153–154).

1.1.1 Struktura subsystému a jeho jednotlivé komponenty

Podkapitola, pokud není uvedeno jinak, čerpá z části oficiální dokumentace kontroléru ONOS (Koshibe a Wang, 2016).

Každý subsystém realizuje určitou službu a je implementován jako kombinace komponent, které se nacházejí ve třech hlavních vrstvách (vizte obrázek 2) – aplikační vrstva, distribuované jádro a vrstva *southbound* protokolů. Komponenty vytvářejí vertikální řez v jednotlivých vrstvách. Každá komponenta subsystému může být identifikována jedním nebo více Java rozhraními. Obrázek 3 znázorňuje vztahy mezi jednotlivými komponentami subsystému. Horní a spodní čárkovaná čára představuje hranice mezi hlavními vrstvami. Hranice jsou tvořeny *southbound*, resp. *northbound* rozhraními. Každý subsystém nemusí nutně obsahovat všechny komponenty, které jsou uvedeny na obrázku 3. Následující odstavce budou věnovány popisu jednotlivých komponent a jejich komunikaci (Rao, 2015).



Obrázek 3: Struktura subsystému

Zdroj: (Koshibe a Wang, 2016)

Provider představuje nejnižší vrstvu softwarového zásobníku ONOS kontroléru. Provideři interagují přímo se sítí pomocí knihoven, které jsou specifické pro jednotlivé komunikační protokoly, a s jádrem skrze rozhraní *ProviderService.* Provideři mají na rozdíl například od aplikačních komponent povědomí o konkrétním použitém komunikačním protokolu. Jsou zodpovědní za interakci se síťovým prostředím pomocí různých řídících a konfiguračních protokolů a za poskytování senzorických dat jádru. Senzorická data jsou specifická pro každou službu jádra, která je požaduje. Provideři také mohou sbírat data od dalších subsystémů a převádět je na senzorická data takového formátu, který vyžaduje daná služba jádra.

Každý provider je spojený s určitým *ProviderId.* Hlavním účelem *ProviderId* je umožnit identifikaci rodiny providerů vzhledem k zařízením a dalším entitám modelu. Díky tomu mohou být entity modelu spojeny s identitou konkrétního providera, zodpovědného za jejich existenci, i poté, co je daný provider odstraněn nebo pozastaven. *ProviderId* obsahuje Uniform Resource Identifier (URI)[6] a díky tomu umožňuje volné párování zařízení s různými rodinami providerů (párování bez přístupu k samotnému providerovi).

Subsystém může být spojen s několika providery. V takovém případě se určuje, který provider bude primární a který sekundární. Primární provider vlastní entity modelu spojené s jeho službou. Sekundární provideři poskytují další informace ve formě překrytí (*overlays*). Pokud by mělo dojít ke konfliktu mezi překrytím a podkladovými informacemi dostane přednost primární provider. Příkladem subsystému, který podporuje více providerů je subsystém zařízení.

[6] Uniform Resource Identifier neboli jednotný identifikátor zdroje je kompaktní sekvence znaků s pevně definovanou strukturou, která poskytuje jednoduchý a rozšiřitelný prostředek k identifikaci abstraktního nebo fyzického zdroje (Berners-Lee et al., 2005).

Manažer (*manager*) se nachází ve vrstvě jádra. Manažer obdrží informace od providerů a dále je distribuuje aplikacím a dalším službám. Poskytuje několik následujících rozhraní:

- *Service* rozhraní představuje z hlediska SDN architektury *northbound* API. Skrze toto rozhraní aplikace nebo další komponenty jádra získávají informace o určitých aspektech stavu sítě.
- Rozhraní *AdminService* také představuje z hlediska SDN architektury *northbound* API. Je zodpovědné za přijímání příkazů sloužících ke správě a jejich následné aplikaci na stav sítě nebo systému.
- Rozhraní *ProviderRegistry* spadá z hlediska SDN architektury pod *southbound* API. Skrze toto rozhraní se jednotliví provideři registrují u manažerů, se kterými chtějí komunikovat.
- Rozhraní *ProviderService* také spadá z hlediska SDN architektury pod *southbound* API. Toto rozhraní je poskytováno providerům registrovaným u manažera. Skrze něj může provider přijímat zprávy od manažera nebo je manažerovi posílat.

Konzumenti rozhraní služeb manažera mohou přijímat zprávy synchronně (dotazováním služby) i asynchronně (pomocí posluchače události). K asynchronnímu přijímání zpráv lze využít rozhraní *ListenerService*, které je součástí rozhraní *Service*. K přijímání určité události je zapotřebí implementovat posluchače dané události. K tomuto účelu slouží rozhraní *EventListener*, které je zapotřebí v aplikaci implementovat.

Úložiště (*store*) se nachází v jádře a je úzce spojeno s určitým manažerem. Úložiště mají za úkol indexování, persistenci a synchronizaci informací přijatých manažerem. Robustnost a konzistence informací napříč několika ONOS instancemi je zajištěna přímou komunikací mezi úložišti.

Aplikace (*app component*) konzumují a manipulují s informacemi agregovanými manažery skrze rozhraní *AdminService a Service*. Aplikace mohou poskytovat širokou škálu funkcionalit od zobrazení síťové topologie ve webovém prohlížeči až po nastavení jednotlivých cest v síti.

Každá aplikace má unikátní *ApplicationId*. Tento identifikátor využívá ONOS ke sledování kontextu spojeného s danou aplikací (například aplikací vložená *flow* pravidla[7]). Pro získání unikátního identifikátoru se aplikace musí zaregistrovat svým jménem u *CoreService*. Jméno aplikace je očekáváno ve tvaru reverzní Domain Name System (DNS) notace.

[7] *Flow* pravidla jsou vkládána do *flow* tabulek, které se nachází na SDN zařízeních. *Flow* pravidlo obsahuje kritéria pro zjištění, zdali příchozí paket spadá pod určité pravidlo, instrukce, co má s paketem SDN zařízení dále provést, čítače paketů a bytů, prioritu a *timeout* po jehož uplynutí je pravidlo odstraněno (Nadeau a Gray, 2013, s. 50–55).

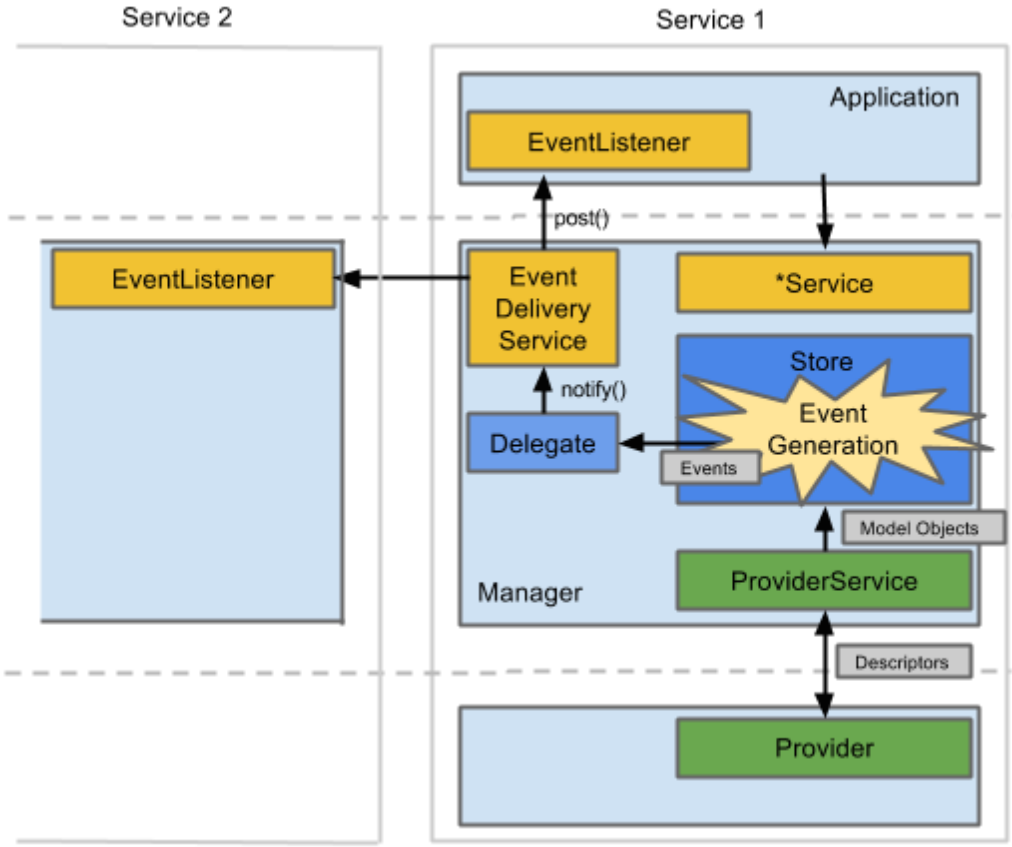
1.1.2 Události a popisy

Podkapitola, pokud není uvedeno jinak, čerpá z části oficiální dokumentace kontroléru ONOS (Koshibe a Wang, 2016).

Události a popisy představují základní jednotky distribuce informace v rámci ONOS kontroléru. Stejně jako služby jsou spojeny se specifickými síťovými prvky a koncepty. Události i popisy jsou po svém vytvoření neměnné. Obrázek 4 znázorňuje vztahy mezi popisy a událostmi, které budou podrobněji popsány v následujících odstavcích, a dává je do kontextu se strukturou subsystému, která byla znázorněna na obrázku 3.

Popisy (*descriptions*) jsou využívány k předávání informací o síťových prvcích skrze *southbound* API. Například popis koncového zařízení (*HostDescription*) obsahuje jeho Media Access Control (MAC) adresu, Internet Protocol (IP) adresu a umístění v síti (např. k jakému SDN zařízení je připojeno). Popisy jsou obvykle tvořeny jedním nebo více modelovými objekty.

Události (*events*) jsou využívány manažery k notifikaci posluchačů o změnách sítě a úložišti k upozornění dalších ONOS instancí v distribuovaném prostředí na změny. Událost se skládá z typu události (např. zařízení bylo odstraněno) a z modelového objektu, který reprezentuje předmět (např. SDN zařízení), který událost vyvolal.



Obrázek 4: Vztah mezi událostmi, popisy a komponentami subsystémů

Zdroj: (Koshibe a Wang, 2016)

Události jsou generovány úložišti na základě vstupu, který úložiště obdrží od manažera. Jakmile je událost vygenerována, je odeslána zaregistrovaným posluchačům pomocí rozhraní *StoreDelegate*. Rozhraní *StoreDelegate* přemístí událost mimo úložiště a služba *EventDeliveryService* zajistí, že událost dorazí pouze registrovaným posluchačům. Obě komponenty (*StoreDelegate, EventDeliveryService*) se nachází v manažerovi. Manažer poskytuje úložišti implementační třídu *StoreDelegate*.

Posluchačem události je libovolná komponenta, která implementuje rozhraní *EventListener*. Rozhraní má své potomky, kteří určují typ události, které bude komponenta naslouchat. Obvykle je třída implementující rozhraní *EventListener* vnitřní třídou manažera nebo aplikace. V reakci na událost je vyvolána určitá jejich služba.

Modelové objekty (*model objects*) reprezentují různé síťové prvky. Struktura modelových objektů je nezávislá na použitém komunikačním protokolu, resp. zařízeních. Modelové objekty jsou vytvářeny jádrem z informací, které se nacházejí v popisech.

1.2 Distribuovaná architektura

SDN architektura je založena na logicky centralizované kontrole sítě. SDN kontroléry společně s nainstalovanými aplikacemi spravují všechna zařízení v síti. Logicky centralizovaná kontrola má řadu výhod, jako je například efektivní správa sítě a schopnost rychlé reakce na dynamické události. Existují dvě varianty, jak dosáhnout logické centralizované kontroly: jeden SDN kontrolér na celou síť a distribuovaný kontrolér[8] (Oktian et al., 2017).

V případě jednoho SDN kontroléru existuje pouze jedna instance, která běží na jednom fyzickém či virtuálním stroji. K této instanci jsou připojena veškerá SDN zařízení v topologii. Dochází tedy k omezení škálovatelnosti a robustnosti sítě. Škálovatelnost znamená, že je možné síť rozšířit, aniž by došlo ke znatelnému zpomalení či dokonce omezení funkčnosti sítě. Jedna instance kontroléru je však limitována svými zdroji a není pro ni reálné zvládnout neomezený počet požadavků od SDN zařízení. Omezení robustnosti znamená, že pokud dojde k výpadku jediného SDN kontroléru, tak ho není čím nahradit. SDN zařízení tedy nemůžou směrovat pakety, pro které nebylo před výpadkem vytvořeno *flow* pravidlo, a může dojít až k celkovému zhroucení sítě. Obě výše zmíněná omezení řeší distribuovaný SDN kontrolér (Oktian et al., 2017).

Jak už bylo zmíněno v úvodu do kapitoly 1, ONOS je již od počátku vyvíjen přímo pro poskytovatele služeb. Nikoli však z hlediska podpory *southbound* protokolů (ONOS je primárně určen pro protokol OpenFlow), ale z hlediska jeho architektury. Architektura kontroléru už od počátků počítá s distribuovaným jádrem a díky tomu umožňuje správu obrovského počtu zařízení (Goransson, Black a Culver, c2017, s. 179–180).

ONOS cluster se skládá z jedné nebo několika instancí neboli uzlů, přičemž každý uzel má svůj unikátní identifikátor. Každý uzel, který je součástí clusteru, má na starost správu své sekce sítě

[8] V tomto případě je řídící vrstva fyzicky distribuována. Nicméně z logického pohledu se stále jedná o jeden centralizovaný kontrolér.

(lokální stav sítě). Lokální stav je distribuován instancí napříč clusterem ve formě událostí. Události jsou sdíleny všemi instancemi, které tvoří cluster. Pomocí distribuce událostí se vytváří globální stav sítě (Koshibe a Olkhovskaya, 2016).

1.2.1 Síťová doména, lokální a globální stav sítě

V distribuovaném ONOS kontroléru má jedna jeho instance na starost pouze část sítě, tzv. doménu. Doména jedné instance je definována všemi SDN zařízeními přímo připojenými k dané instanci a všemi koncovými zařízeními, které jsou k nim připojeny. Pokud jsou zařízení přímo připojena k více instancím kontroléru, tak je doména tvořena pouze zařízeními, pro které je daná instance tzv. *masterem* (Koshibe a Olkhovskaya, 2016; Oktian et al., 2017).

Lokální stav sítě reprezentuje současný stav domény jedné instance a je uložen v jejím úložišti. ONOS má díky tomuto stavu povědomí o všech událostech v doméně, například připojení/odpojení SDN zařízení, porucha propojení atd. Lokální stav se dělí na dvě kategorie: statický, pod nějž spadají informace jejichž modifikace není tak častá, a dynamický, který se skládá z informací, které se často mění (Oktian et al., 2017).

ONOS kontrolér také udržuje globální stav sítě. Globální stav sítě slouží ke správě sítě jako celku a sdílení lokálních stavů sítě napříč clusterem. Reprezentuje stav sítě napříč doménami. Ke konstrukci globálního stavu je zapotřebí, aby každá instance distribuovala svůj lokální stav. Poté jsou všechny lokální stavy spojeny a je vytvořen jeden globální (Oktian et al., 2017; Berde et al., 2014).

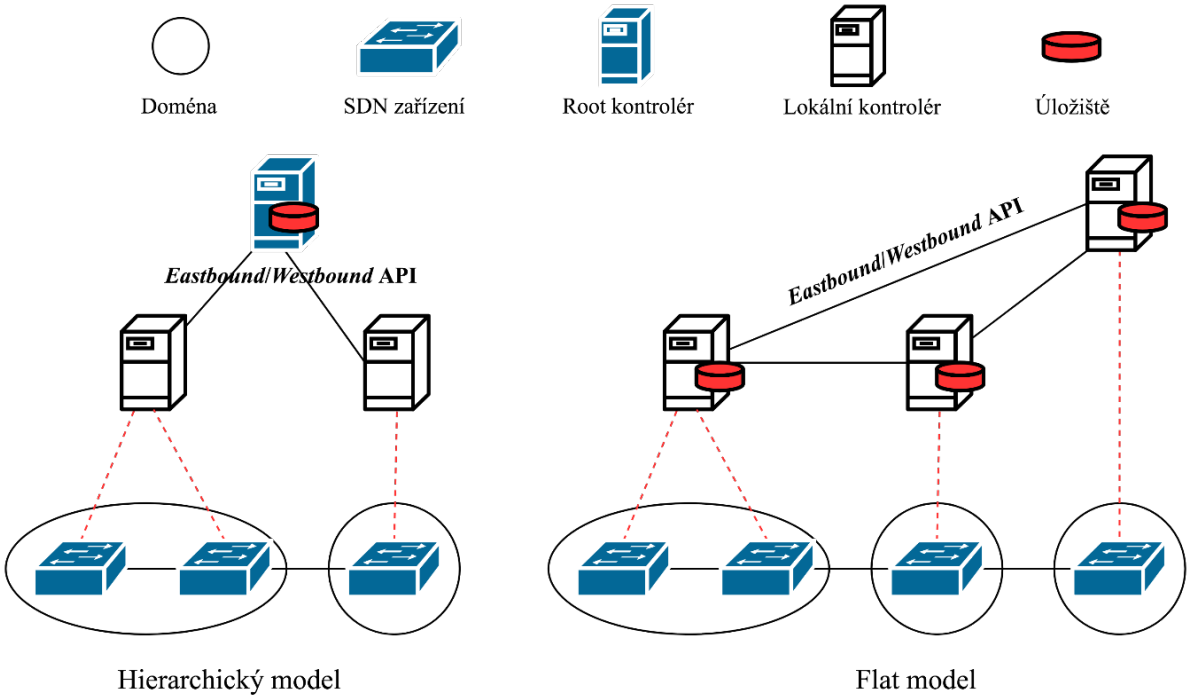
1.2.2 Organizace clusteru

Podkapitola, pokud není uvedeno jinak, čerpá z článku o distribuovaných SDN kontrolérech (Oktian et al., 2017).

Organizace clusteru je u kontroléru ONOS postavena na *flat* modelu (znázorněn na obrázku 5), který je také známý pod pojmem horizontální architektura. U tohoto modelu je třeba zajistit správné uspořádání jednotlivých událostí, které mohou být libovolně vysílány z více instancí, a vyřešit poruchy, ke kterým může docházet při všesměrovém vysílání událostí. V následujících odstavcích budou popsány možné jednotlivé role instancí, *flat* model, princip komunikace mezi instancemi a způsob řešení problémů *flat* modelu ONOS kontrolérem.

Instance clusteru může mít vzhledem k SDN zařízení tři role: *none, standby, master*. Každé zařízení může mít pouze jednoho *mastera*. Role *none* znamená, že instance nemusí mít o zařízení povědomí a nemůže s ním komunikovat. Role *standby* znamená, že instance má povědomí o zařízení. Může zjišťovat jeho stav, ale nemá právo zápisu. Role *master* znamená, že instance ví o daném zařízení a má nad ním plnou kontrolu (právo čtení i zápisu). Role zařízení vzhledem k instanci může být určena automaticky (instance, která jako první naváže spojení se zařízením a zároveň zjistí, že zařízení nemá *mastera* se stává *masterem*) nebo administrativně. Každá instance začíná vzhledem k zařízení na roli *none.* Pokud zařízení ztratí spojení s *master* instancí, tak je zvolena nová *master* instance. Ta je zvolena z instancí, které jsou vzhledem k zařízení

v roli *standby*. Všechny instance přímo připojené k zařízení v roli *standby* jsou seřazeny podle svého unikátního identifikátoru v rámci clusteru. První v pořadí je zvolena jako *master*. Role ONOS instancí vzhledem k zařízením jsou mapovány na OpenFlow role: *master* na *OFPCR_ROLE_MASTER* a *standby* na *OFPCR_ROLE_SLAVE* (Koshibe a Olkhovskaya, 2016; OpenFlow Switch Specification, 2015).



Obrázek 5: Hierarchický model versus *flat* model

Zdroj: zpracováno podle (Oktian et al., 2017)

Princip *flat* modelu spočívá v tom, že instance nejsou organizovány v hierarchické struktuře. Všechny instance jsou si rovny. Všechny mají úložiště, kde si udržují globální stav sítě, a *eastbound/westbound* API, skrze které komunikují s ostatními instancemi. Předtím, než může instance zkonstruovat globální stav sítě, musí získat lokální stavy sítě od všech ostatních instancí v clusteru. Také veškeré změny, které nastanou v rámci domény instance, musí být sdíleny s ostatními instancemi. Když je synchronizace dokončena, tak by měly mít všechny instance v clusteru stejný globální stav sítě. Za předpokladu, že nedošlo k chybě.

K řešení problému s uspořádáním událostí využívá ONOS kontrolér logická časová razítka. Každá událost je před posláním dalším instancím v clusteru opatřena logickým časovým razítkem. Při příjmu se nejprve zkontroluje logické časové razítko. Pokud už instance přijala zprávu o události týkající se daného síťového prvku s vyšším logickým časovým razítkem, tak je zpráva zahozena. Logické časové razítko je n-tice, která se skládá z *term_number* a *event_number*. *Term_number* je inkrementován pokaždé, když je zvolena nová *master* instance pro dané zařízení. Inkrementaci předchází proces volby nové *master* instance pro dané zařízení. *Event_number* je inkrementován pokaždé, když nastane nějaká událost týkající se daného zařízení. Při porovnávání se nejprve porovnává *term_number* a teprve při shodě *event_number*.

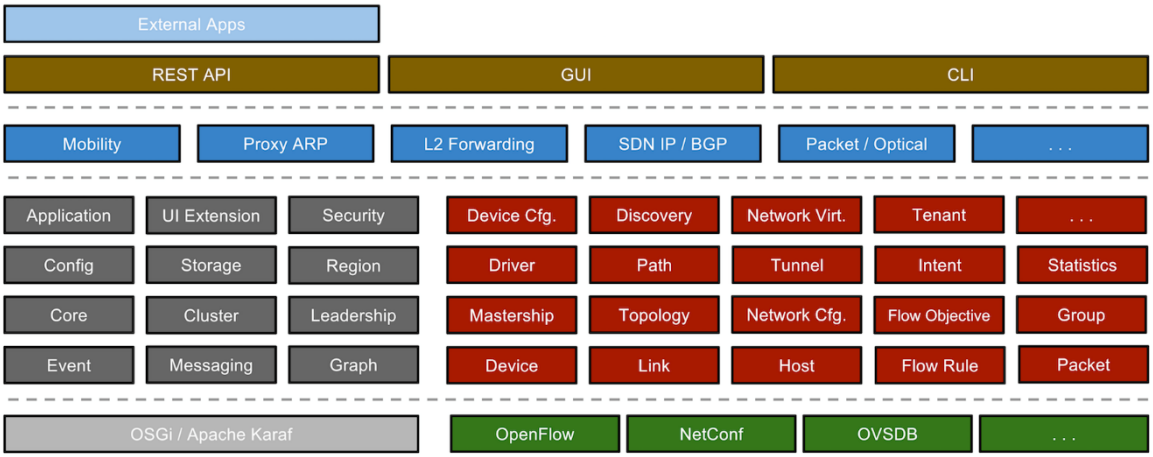
Porovnávají se pouze události, které byly vyvolány stejným zařízením. Události vyvolané různými zařízeními jsou na sobě nezávislé (Jampani a Jianwei, 2016).

Kromě uspořádání událostí je zapotřebí také vyřešit, jakým způsobem budou řešeny problémy s potenciálním nedoručením či ztrátou události. K nedoručení události může dojít v případě, kdy dojde k selhání některé instance během její distribuce a událost se tím pádem nedostane ke všem ostatním instancím v clusteru. Nedoručení události řeší ONOS kontrolér pomocí mechanismu založeném na Gossip protokolu – instance si v pravidelném intervalu náhodně vybírá jinou instanci v clusteru a provádí synchronizaci svého pohledu na síť s jejím. Pokud jedna z instancí zjistí, že má druhá novější informace, tak aktualizuje svůj pohled na síť. Ke ztrátě události může dojít v případě, že zařízení vyvolá událost, ale jeho *master* instance selže dříve, než ji zařízení o této události uvědomí. Ztráta události se řeší tak, že se instance periodicky dotazuje všech zařízení, pro která je *masterem*. Pokud instance detekuje, že stav zařízení je jiný, než má uložený v lokálním stavu sítě, tak svůj lokální stav okamžitě aktualizuje. Tuto aktualizaci poté distribuuje ostatním instancím v clusteru (Jampani a Jianwei, 2016).

2 HLAVNÍ SUBSYSTÉMY ONOS KONTROLÉRU

Dříve než budou popsány hlavní subsystémy, je třeba vysvětlit termín aplikace z pohledu ONOS kontroléru, neboť bude frekventovaně používán v následujících kapitolách. Z pohledu ONOS kontroléru existují dva druhy aplikací:

- ONOS aplikace, které se nacházejí na aplikační vrstvě ONOS kontroléru. Tyto aplikace jsou spouštěny a spravovány ONOS kontrolérem. Speciální typem ONOS aplikace je tzv. modul. Termínem modul bude v rámci diplomové práce označována ONOS aplikace, která interaguje zprostředkovaně se sítí skrze APIs. Příkladem takové interakce může být vkládání *flow* pravidel na SDN zařízení.
- Externí aplikace, které se nacházejí mimo ONOS kontrolér. Ke komunikaci s ONOS kontrolérem a ONOS aplikacemi je využíváno například Representational State Transfer (REST) API.



Obrázek 6: Subsystémy ONOS kontroléru

Zdroj: (Koshibe a Wang, 2016)

Jak již bylo zmíněno v kapitole 1.1, na ONOS architekturu lze také nahlížet jako na třístupňovou kolekci subsystémů, kde každý subsystém realizuje určitou službu a je implementován jako kombinace komponent. Obrázek 6 zobrazuje různé subsystémy, které jsou v současné době součástí ONOS kontroléru. Z obrázku je patrné, že termínem subsystém se v oficiální dokumentaci označuje téměř vše, co je součástí ONOS kontroléru. V této kapitole budou však představeny pouze hlavní subsystémy ONOS kontroléru spadající do jádra (červená[9] a šedá barva) a vrstvy *southbound* protokolů (zelená barva). Subsystémům znázorněným tmavě modrou barvou, které jsou také označovány jako ONOS aplikace, je věnována kapitola 4 (Koshibe a Wang, 2016).

[9] Červenou barvou jsou znázorněny subsystémy, které slouží k interakci s SDN sítí.

2.1 Konstrukce stavu sítě

Podkapitola, pokud není uvedeno jinak, čerpá z části oficiální dokumentace ONOS kontroléru (Koshibe et al., 2016).

Stav sítě je klíčovou informací, která je udržována řídící vrstvou. Řídící vrstva musí shromáždit veškeré informace o spravované topologii a poskytnout je ONOS aplikacím, které se nacházejí na aplikační vrstvě (obrázek 6 tmavě modrá barva). Navíc musí být ONOS aplikace odstíněny od specifického komunikačního protokolu, skrze který byly informace získány. Důvodem je zachování přenositelnosti a rozšiřitelnosti ONOS aplikací.

Topologie udržovaná ONOS kontrolérem je nezávislá na použitém komunikačním *southbound* protokolu (např. OpenFlow). Je konstruována pomocí dvou vzájemně se doplňujících mechanismů – zjišťování a následná konfigurace sítě. První mechanismus využívá konkrétní síťový protokol, který ONOS kontroléru umožňuje identifikovat, kde se zařízení nachází a/nebo jeho vlastnosti. Identifikace je prováděna proaktivně[10]. Druhý mechanismus umožňuje ONOS aplikacím či správcům sítě konfigurovat očekávanou topologii nebo poskytovat „rady“ ohledně síťových komponent, které nemohou být objeveny typickými prostředky.

V rámci této kapitoly bude vysvětleno, jakým způsobem ONOS kontrolér reprezentuje síť, resp. její prvky a stav, jakým způsobem je vytvářena topologie, resp. pohled na topologii a jak funguje subsystém pro konfiguraci sítě.

2.1.1 Síťová reprezentace

ONOS kontrolér udržuje ke každému síťovému prvku a jejich stavům dva druhy reprezentací – nezávislou na použitém protokolu a specifickou pro použitý protokol. První druh reprezentace může být převeden na druhý a naopak. Převod mezi jednotlivými druhy reprezentací se provádí pomocí popisů. Reprezentace prvního druhu jsou konstruovány jádrem ONOS kontroléru a jsou označovány jako modelové objekty. Modelové objekty jsou tím, co vrstva jádra poskytuje ONOS aplikacím. Reprezentace druhého jsou konstruovány příslušným providerem.

Modelové objekty jsou definovány pomocí rozhraní, která ONOS kontrolér implementuje. Existují celkem tři různé skupiny modelových objektů:

- Modelové objekty síťové topologie jsou analogické s prvky grafů, neboť ONOS reprezentuje síť jako orientovaný graf. Mezi modelové objekty síťové topologie patří například: *Device* reprezentující konkrétní prvek síťové infrastruktury (vnitřní vrchol grafu), *Port* reprezentující síťové rozhraní (koncový bod hrany grafu) a *Host* reprezentující koncové zařízení v síti (vnější vrchol grafu).
- Modelové objekty sloužící k řízení sítě. Na ONOS aplikační vrstvě jsou direktivy pro síť vyjádřeny ve formě *flow* pravidel. Každé *flow* pravidlo se skládá z kritérií (*Criteria*)

[10] Zařízení jsou okamžitě po navázání spojení s ONOS kontrolérem přidána do globálního pohledu na topologii.

a způsobu zpracování (*Treatment*). *Flow* pravidla na ONOS aplikační vrstvě nejsou totéž, co *flow* pravidla u OpenFlow protokolu. ONOS kontrolér například nepodporuje nastavení typu služby (*type of service*) v hlavičce paketu. Mezi modelové objekty pro řízení sítě patří například: *FlowEntry* reprezentující *flow* pravidlo v tabulce určitého SDN zařízení a *Intent* umožňující ONOS aplikaci specifikovat, co se má provést, nehledě na to, jakým způsobem to bude provedeno.

- Modelové objekty reprezentující síťový provoz jsou analogické s OpenFlow *PacketIn* a *PacketOut*. Existují tedy pouze dva druhy objektů reprezentující síťový provoz: *OutboundPacket* reprezentující paket vyslaný ONOS kontrolérem do sítě a *InboundPacket* reprezentující paket poslaný ONOS kontroléru SDN zařízením.

Některé modelové objekty jsou závislé na existenci dalších. Například instance třídy *Port* nemůže být vytvořena bez instance třídy *Device*. Instance tříd reprezentující topologii a spojení zase nemůžou být vytvořeny bez portů. Instance třídy *Device* je jedinou třídou, jejíž instanci je možné vytvořit bez dalších instancí tříd. Jedná se tedy o nezávislou třídu.

2.1.2 Zjišťování topologie sítě

ONOS kontrolér dokáže získat kompletní pohled na topologii pouze s malým zásahem správce sítě. Jediné, co správce sítě musí nakonfigurovat, je přímé spojení SDN zařízení s ONOS kontrolérem. ONOS kontrolér přidává SDN zařízení do svého pohledu na topologii v okamžiku, kdy s ním SDN zařízení naváže spojení. V závislosti na použitém protokolu k vytvoření spojení mohou být ONOS kontroléru poskytnuty ještě další informace (např. počet dostupných portů SDN zařízení). Informace ohledně vlastností SDN zařízení jsou ONOS kontroléru předávány při nastavení parametrů komunikačního kanálu mezi ONOS kontrolérem a SDN zařízením – tento proces se označuje jako *handshake*.

V případě zjišťování spojení a koncových zařízení se ONOS kontrolér spoléhá na schopnost získávání informací od SDN zařízení a jejich konfiguraci. V této podkapitole bude vysvětleno, jakým způsobem ONOS kontrolér získává informace o koncových zařízeních a spojeních, když už má nad SDN zařízeními plnou kontrolu. Plná kontrola znamená, že s SDN zařízením bylo navázáno spojení a ONOS kontrolér si ho přidal do svého pohledu na topologii.

Zjišťování spojení v topologii provádí subsystém spojení. Ten je propojen se subsystémem zařízení prostřednictvím providera *LLDPLinkProvider*. *LLDPLinkProvider* se přihlásí k odběru událostí týkajících se SDN zařízení a průběžně se také neustále dotazuje na informace o jejich stavu. Dále pro každé objevené SDN zařízení také alokuje objekt *LinkDiscovery*, který implementuje mechanismus pro zjišťování spojení pomocí Link Layer Discovery Protocol (LLDP) a Broadcast Domain Discovery Protocol (BDDP) protokolu.

Každá instance *LinkDiscovery* v pevných časových intervalech odesílá skrze SDN zařízení, kterému náleží, LLDP a BDDP pakety. Pakety jsou odeslány na všechny porty SDN zařízení. Paket zjišťovacího protokolu obsahuje unikátní identifikátor odesilatele společně s portem. V momentě, kdy je LLDP paket přijat na druhé straně spojení, tak je odeslán SDN zařízením ONOS kontroléru. Ten jeho zpracování deleguje na příslušnou instanci *LinkDiscovery* náležící danému

SDN zařízení. Instance *LinkDiscovery* zjistí odesilatele a příjemce paketu a porty, kterými jsou spojeni. Tím je detekováno, že mezi odesilatelem a příjemcem existuje přímé spojení. V tomto spojení je odesilatel počátečním bodem a příjemce koncovým. Instance *LinkDiscovery* nepředpokládá, že by spojení bylo obousměrné. Stejný princip, jako byl popsán výše, se používá i pro zjištění spojení v opačném směru. LLDP protokol se používá pouze ke zjišťování přímých spojení. Jeho pakety nejsou dále směrovány.

Ke zjišťování spojení mezi SDN ostrovy[11] a tradičními sítěmi se používá BDDP[12] protokol. Stejně jako ONOS kontrolér ani tradiční zařízení dále nesměruje přijaté LLDP pakety. Mezi jednotlivými SDN ostrovy propojenými pomocí tradičních zařízení by kvůli tomu pro ONOS kontrolér nebylo možné zjistit spojení. ONOS kontrolér by si mylně vytvořil dva pohledy na topologii, které by spravoval zvlášť. BDDP pakety však tradiční zařízení nezahazuje ani je nijak nemodifikuje. BDDP paket projde skrze všechna tradiční zařízení až k dalšímu SDN zařízení spravovanému ONOS kontrolérem. Z tohoto důvodu se jeví ONOS kontroléru spojení mezi dvěma SDN ostrovy jako spojení pouze s jedním skokem.

Zjišťování koncových zařízení provádí subsystém zařízení pomocí Address Resolution Protocol (ARP) a Dynamic Host Configuration Protocol (DHCP) zpráv. Tyto zprávy jsou posílány ONOS kontroléru (*PacketIn*). Zprávy obsahují port a unikátní identifikátor SDN zařízení, ke kterému je koncové zařízení připojeno. Tento mechanismus implementuje *HostLocationProvider* podobně jako *LLDPLinkProvider.*

2.1.3 Subsystém síťové konfigurace

Subsystém síťové konfigurace umožňuje ONOS aplikacím či správci, pomocí REST API, vkládat informace o atributech a konfiguracích různých síťových prvků. Tyto informace jsou vkládány do modelových objektů sloužících pro reprezentaci sítě. Například je možné nastavit typy a unikátní identifikátory portů a zařízení nebo zda bude či nebude konkrétní síťový prvek zahrnut do topologie. Je možné dokonce konfigurovat prvek sítě, který ještě neexistuje v síťové reprezentaci (ještě nebyl objeven). ONOS aplikace tedy může jednak poskytovat „rady“ ohledně prvků, které ještě nebyly objeveny, stejně jako modifikovat atributy již objevených prvků.

Subsystém síťové konfigurace také slouží jako prostředník mezi síťovou reprezentací a prostředky sloužícími k její konfiguraci. V současné době je preferovaným prostředkem pro konfiguraci dokument ve formátu JavaScript Object Notation (JSON). Subsystém síťové konfigurace aplikuje konfiguraci ve formátu JSON na síťovou reprezentaci.

[11] SDN ostrov je část sítě, která je řízena SDN kontrolérem. SDN ostrovy mohou být mezi sebou propojeny jedním, ale i několika tradičními zařízeními (Koshibe et al., 2016).
[12] BDDP protokol lze použít pouze v případě, že se SDN ostrovy a tradiční síťová zařízení nacházejí ve stejné broadcastové doméně (Hong et al., 2015).

Subsystém síťové konfigurace lze použít ke konfiguraci libovolných síťových prvků (objektů). Nicméně, aby byl objekt konfigurovatelný, je nutné implementovat následující komponenty:

- Subjekt je Java objekt, který slouží k unikátní identifikaci síťového prvku. Obsahuje také odkaz na objekt, který je předmětem konfigurace.
- Konfigurační třídu implementující metody, které umožňují nastavit a získat atributy daného subjektu.
- *SubjectFactory* sloužící ke spojení subjektu s jeho klíčem[13] a ke generování objektů, které reprezentují subjekty.
- *ConfigFactory* sloužící ke spojení subjektu s jeho konfigurací a ke generování objektů, které reprezentují konfiguraci subjektu.
- *ConfigOperator* sloužící ke sloučení informací z různých zdrojů a ke konverzi informací mezi zdroji. Například *BasicLinkOperator* sloužící k definici metod pro sloučení obsahu *BasicLinkConfig* a *LinkDescription* a ke konverzi *Link* a *BasicLinkConfig* objektů na objekty typu *LinkDescription*. S objekty typu *LinkDescription* dokáže pracovat jádro ONOS kontroléru.

Pro standardní síťové prvky má ONOS kontrolér tyto komponenty implementovány. Jejich konfiguraci zpřístupňuje služba pro konfiguraci sítě skrze REST API.

2.2 Subsystém zařízení

Podkapitola, pokud není uvedeno jinak, čerpá z části oficiální dokumentace ONOS kontroléru (Koshibe a Shakil, 2017).

Subsystém zařízení je zodpovědný za zjišťování a neustálé sledování zařízení, která tvoří síť, a umožňuje ONOS aplikacím a správcům sítě jejich řízení. Většina hlavních subsystémů jádra ONOS kontroléru využívá modelových objektů vytvořených a spravovaných právě tímto subsystémem (např. *Device* a *Port*) a/nebo některého z jeho providerů k interakci se sítí.

Architektura subsystému zařízení se řídí obecnou architekturou struktury subsystémů, která byla popsána v kapitole 1.1.1, a obsahuje následující komponenty:

- Manažera zařízení, který představuje implementaci manažerské komponenty. Je schopný propojit více providerů prostřednictvím *DeviceProviderService* rozhraní a více posluchačů skrze rozhraní *DeviceService*.
- Providery zařízení, kteří představují implementaci komponenty providera. Každý z providerů obsahuje knihovny pro podporu různých komunikačních protokolů nebo prostředků, jak se spojit se sítí.

[13] Klíč je textová reprezentace názvu subjektu.

- Úložiště zařízení sloužící k uložení modelových objektů zařízení a ke generování událostí týkajících se jednotlivých zařízení.

Jak již bylo uvedeno v kapitole 1.1.2, od vrstvy jádra a výše se už používají výlučně modelové objekty nezávislé na protokolu. Naopak *southbound* API, ve kterém se nachází konkrétní provideři, používá objekty závislé na protokolu. Je tedy zapotřebí transformace protokolově závislých objektů na nezávislé. Transformace je prováděna v jádře kontroléru ONOS. V tabulce 1 se nachází příklad mapování jednotlivých modelových objektů na objekty používané OpenFlow providerem zařízení.

Tabulka 1: Mapování modelových objektů na objekty používané providery zařízení

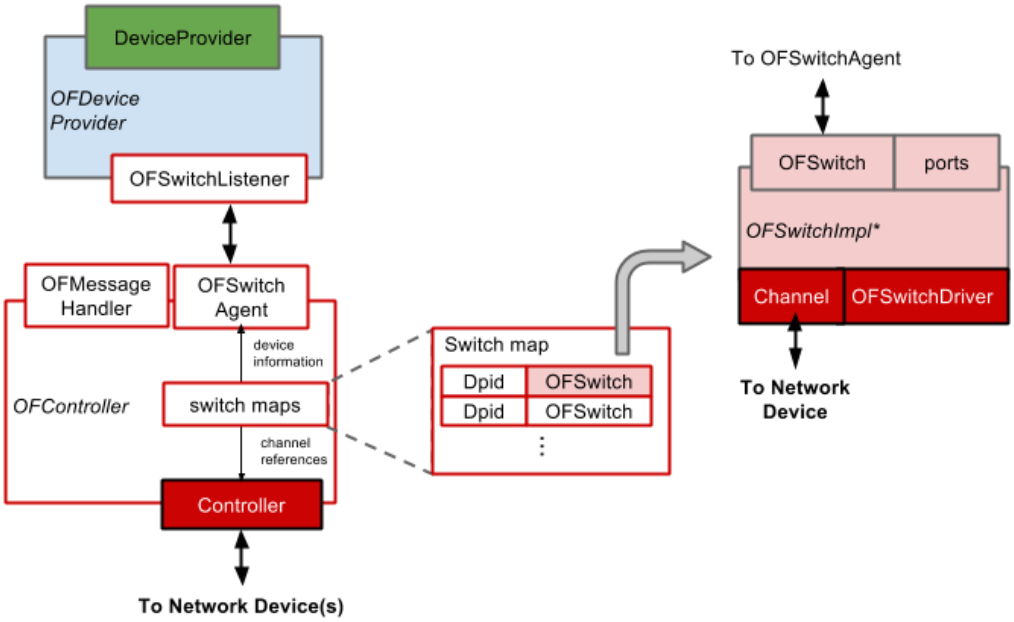
Manažer zařízení	**Provider zařízení pro protokol OpenFlow**
Device	OpenFlowSwitch
DeviceId/ElementId	Dpid
MastershipRole	RoleState
Port	OFPortDesc

Zdroj: (Koshibe a Shakil, 2017)

2.2.1 OpenFlow subsystém

OpenFlow *southbound* API se skládá ze dvou komponent: OpenFlow providera a ovladačů pro OpenFlow zařízení. Ačkoli se z pohledu ONOS kontroléru nejedná o subsystém, je v oficiální dokumentaci soubor výše zmíněných komponent jako subsystém označován. OpenFlow subsystém implementuje chování OpenFlow kontroléru pomocí OpenFlowJ Loxi. OpenFlowJ Loxi poskytuje rozhraní nezávislé na používané verzi protokolu OpenFlow umožňující například: tvorbu *flow* pravidel, vkládání *flow* pravidel do *flow* tabulky zařízení a směrování paketů. Nevýhodou OpenFlowJ Loxi je, že podporuje pouze starší verze OpenFlow protokolu: 1.0 a 1.3. V současné době je volně dostupná již verze 1.5.1. Existuje i verze 1.6, ale ta je určena pouze pro členy ONF.

Obrázek 7 znázorňuje architekturu subsystému OpenFlow. Modrý blok představuje provider komponentu a zelený blok představuje rozhraní mezi providerem a manažerem (vizte obrázek 3). Červené, růžové a červeně ohraničené bloky představují složky bloku označeného jako „Protocols“ na obrázku 3, který znázorňuje strukturu subsystému. Červené a růžové bloky se přímo podílejí na komunikaci s jednotlivými zařízeními skrze Transmission Control Protocol (TCP) spojení.



Obrázek 7: Architektura subsystému OpenFlow

Zdroj: (Koshibe a Shakil, 2017)

OpenFlow kontrolér (na obrázku 7 znázorněn jako „OFController“) umožňuje používání OpenFlow funkcí a skrze tyto funkce řízení OpenFlow zařízení. OpenFlow kontrolér uchovává mapování Datapath ID (DPID)[14] na instance třídy *OpenFlowSwitch* (na obrázku 7 znázorněna jako „OFSwitchImpl*“) a generuje OpenFlow události, k jejichž odběru se mohou přihlásit provideři zařízení. Například může provider naslouchat, zdali nedošlo k odpojení zařízení. Je také zodpovědný za vytváření a správu komunikačních kanálů pro každou instanci třídy *OpenFlowSwitch*. Spojení s jednotlivými zařízení je navázáno kontrolérem (na obrázku 7 znázorněn jako „Controller“) a stav každého připojeného zařízení je sledován tzv. agentem (na obrázku 7 znázorněn jako „OpenFlowSwitchAgent“). Spojení je navázáno tak, že kontrolér přiřadí OpenFlow TCP kanál příchozímu spojení. OpenFlow TCP kanál je přiřazen tak, že je vytvořena instance třídy *OpenFlowSwitch* s odkazem na TCP spojení.

Instancemi třídy *OpenFlowSwitch* reprezentuje OpenFlow subsystém síťová zařízení. Každá instance obsahuje informace o portech, unikátní identifikátor, informace o schopnostech zařízení a odkaz na skutečné TCP spojení s daným zařízením. Každá instance má dva typy rozhraní: *OpenFlowSwitch* a *OpenFlowSwitchDriver* (na obrázku 7 znázorněny jako „OFSwitch“ a „OFSwitchDriver“). První typ rozhraní umožňuje providerům a nepřímo i celému ONOS kontroléru interakci s instancemi třídy *OpenFlowSwitch*. Skrze instance třídy *OpenFlowSwitch* jsou prováděny interakce s fyzickými zařízeními. Druhý typ rozhraní se zabývá detaily protokolu OpenFlow a nevyžaduje téměř žádný zásah od zbytku systému.

[14] DPID je unikátní identifikátor konkrétní *pipeline*, která je spravována OpenFlow kontrolérem. Část 64bitového identifikátoru je tvořena MAC adresou a část určuje programátor dané *pipeline*. *Pipeline* znamená zřetězené zpracování paketů. Paket může při zpracování projít pouze jednou nebo více *flow* tabulkami (OpenFlow Switch Specification, 2015).

Instance třídy *OpenFlowSwitch* má několik typů stavů, ve kterých se může nacházet. Na každý stav je navázáno nějaké chování, například jaký typ řídících zpráv od kontroléru zařízení očekává nebo mu zasílá. Dva primární typy stavů spojené se zařízením jsou stav kanálu spojení s kontrolérem a role instance ONOS kontroléru vzhledem k zařízení. Stav kanálu spojení je implementován jako konečný stavový automat. Přechody mezi stavy jsou řízeny příjmem různých zpráv během nastavení parametrů komunikačního kanálu. Role instance vzhledem k zařízení byly popsány v kapitole 1.2.

2.3 Subsystém ovladačů zařízení

Podkapitola, pokud není uvedeno jinak, čerpá z části oficiální dokumentace ONOS kontroléru (Vachuska a Kandi, 2016).

Z pohledu ONOS kontroléru reprezentuje ovladač celou rodinu zařízení nebo konkrétní zařízení. Účelem subsystému ovladačů zařízení je izolovat kód specifický pro dané zařízení, aby nedošlo k jeho rozšíření do celého systému. Subsystém ovladačů zařízení umožňuje takový kód obsáhnout a zpřístupnit ho ONOS aplikacím prostřednictvím dobře definovaných abstrakcí, které jsou nezávislé na zařízení a komunikačním protokolu. Dále, vzhledem k tomu, že zařízení jsou vydávána a inovována v odlišných cyklech než ostatní platformy pro řízení sítě, poskytuje mechanismus umožňující asynchronní a dynamické načítání kódu specifického pro zařízení.

Kvůli tomu, že různá zařízení mohou nabízet stejné funkcionality, ale zároveň poskytovat i unikátní, není architektura ovladačů ONOS kontroléru monolitická[15]. Namísto toho architektura podporuje segmentaci různých aspektů chování. Segmentace umožňuje selektivní podporu funkcionalit a jejich sdílení (pomocí dědičnosti) a přidávání z různých zdrojů.

Zařízení se mohou lišit jak ve způsobu interakce s nimi, tak i z hlediska poskytovaných funkcionalit. Různé aspekty chování z hlediska funkcionalit a interakce se zařízením jsou implementovány jako odvozeniny z *Behavior*, resp. *HandlerBehavior* rozhraní. Implementované odvozeniny výše zmíněných rozhraní jsou součástí konkrétního ovladače zařízení a jsou dostupné skrze subsystém zařízení. V implementaci těchto rozhraní se nachází kód specifický pro dané zařízení a protokol. Základní implementace těchto rozhraní je poskytována subsystémem ovladačů zařízení.

Definice chování by měla být zaměřena na charakterizaci specifického aspektu celkových schopností zařízení, například Virtual Local Area Network (VLAN) konfigurace. Chování by nemělo pokrývat více funkcionalit ani pouze část funkcionality. Korektně definované chování musí zařízení buď podporovat jako celek nebo ho nepodporovat vůbec. Korektně definovaná chování snižují komplexnost ONOS aplikací a usnadňují programování.

[15] Monolitická architektura znamená, že je software navržen tak, aby byl samostatný. Jedná se o architekturu složenou z úzce vázaných komponent. V architektuře složené z úzce vázaných komponent musí být každá komponenta a s ní spojené komponenty k dispozici, aby mohl být kód prováděn nebo kompilován (Rouse a Wigmore, 2016).

Subsystém ovladačů zařízení se skládá z následujících komponent:

- Providera zařízení poskytujícího ovladače zařízení společně s jejich chováním.
- *DriverAdminService* umožňující ONOS aplikacím správu providerů zařízení.
- *DriverService* umožňující ONOS aplikacím a dalším ONOS subsystémům vyhledat konkrétní ovladač zařízení podle: výrobce, podporovaných chování, jména zařízení a unikátního identifikátoru zařízení. Všechny tyto atributy jsou stejně jako chování součástí ovladače zařízení.

2.4 Intent framework

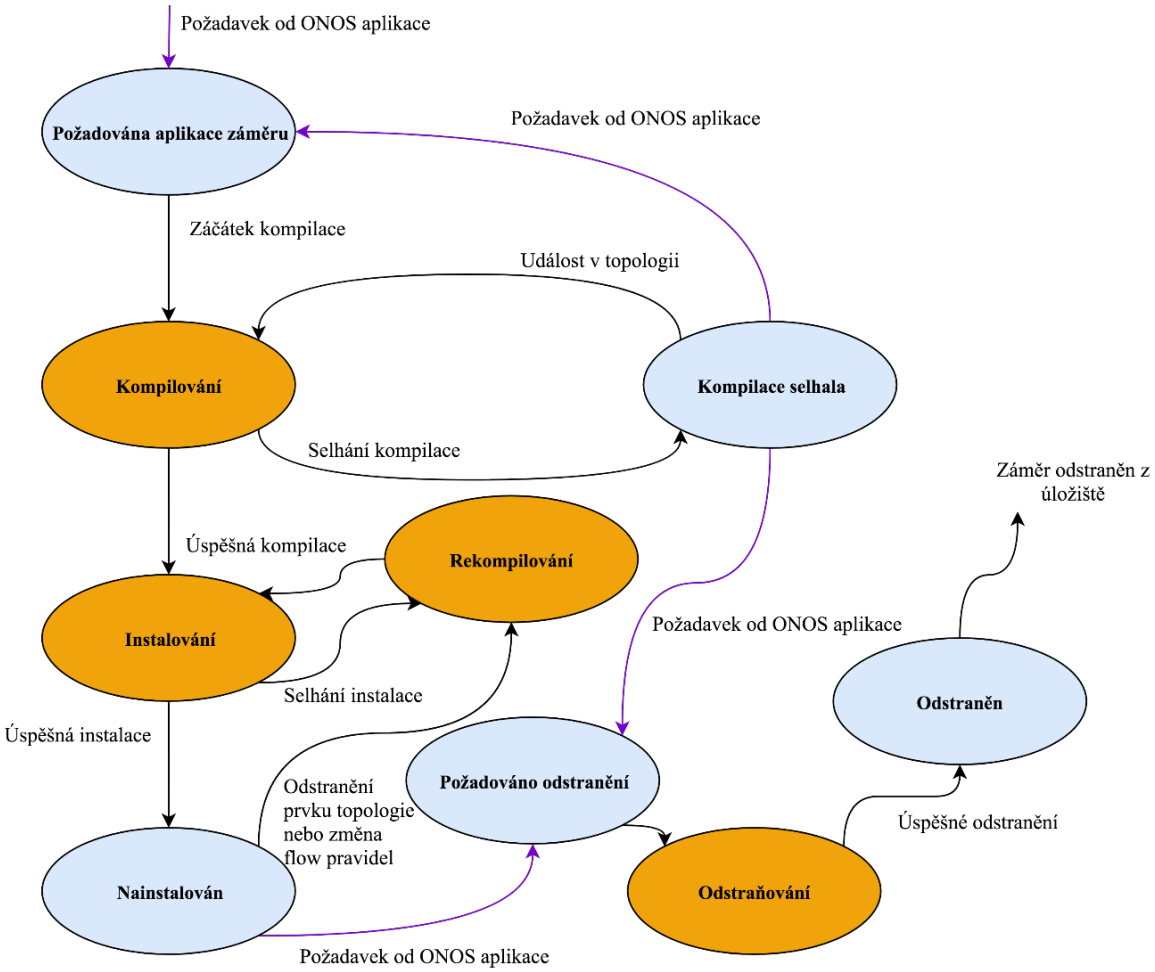
Intent framework je subsystém, který umožňuje ONOS aplikacím specifikovat požadavky na řízení sítě pomocí síťových politik. Síťové politiky představují jednoduchá pravidla, která se skládají z podmínky a akcí. Příkladem síťové politiky může být blokování komunikace, pokud pochází z určitého adresního rozsahu. Direktivy založené na síťových politikách jsou označovány jako záměry (*intent*). Představují nejvyšší úroveň abstrakce. Správci sítí a vývojáři aplikací se díky nim mohou soustředit na to, co je třeba udělat a nikoli na to, jakým způsobem to udělat. Příkladem záměru může být požadavek na vytvoření obousměrného point-to-point spojení mezi dvěma koncovými zařízení. Správce sítě či vývojář aplikace pouze specifikuje, mezi kterými zařízeními se má point-to-point spojení vytvořit. Neřeší, jakým způsobem toho bude dosaženo (Rouse, 2011a; Sanvito et al., 2018).

Z pohledu ONOS kontroléru je záměr neměnný objekt[16], který popisuje požadavek ONOS aplikace na změnu chování sítě. ONOS kontrolér nabízí několik typů záměrů, přičemž každý z nich je dodáván společně s vlastním kompilátorem. Záměry jsou vázány na konkrétní koncová zařízení nebo na specifickou podmnožinu provozu (určena pomocí selektoru – množina specifických hodnot v hlavičce paketu) a spojeny s konkrétní množinou akcí, která je aplikována na všechny pakety, které záměr zpracovává. Některé typy záměrů navíc umožňují definovat sadu omezení pro kompilátor. Například je možné požadovat, aby výsledná cesta procházela určitými uzly či rezervovat určitou šířku pásma (Koshibe a Hart, 2016a).

Na obrázku 8 jsou znázorněny přechody mezi stavy procesu aplikace záměru na síť. Stavy znázorněné oranžovou barvu jsou přechodné a předpokládá se, že budou trvat pouze krátkou dobu. U stavů znázorněných modrou barvou se předpokládá, že v nich záměr může strávit delší dobu. Aplikace záměru na síť probíhá tak, že ONOS aplikace pošle záměr označený jejím unikátním identifikátorem (*ApplicationId*) jádru ONOS kontroléru, které ho přijme a přiřadí mu vlastní unikátní identifikátor (*IntentId*). Následuje fáze kompilace, při níž je záměr přeložen do souboru nízko-úrovňových operací. Například vložení, modifikace, smazání *flow* pravidla. Při kompilaci jádro ONOS kontroléru rozhoduje o tom, jak nejlépe záměr realizovat nebo dokonce o tom, zdali je jeho provedení vůbec možné. Pokud je záměr vyhodnocen jako neproveditelný, přechází do stavu kompilace selhala. V tomto stavu setrvává dokud, nenastane změna

[16] Neměnný objekt je takový objekt, u kterého není možné po jeho vytvoření měnit jeho atributy (Valkovič, c2019).

v topologii nebo není explicitně vyžadován opětovný pokus o aplikaci záměru ONOS aplikací. Pokud byla kompilace úspěšná, následuje fáze instalace záměru. Během instalace jsou provedeny všechny nízko-úrovňové operace, na které byl záměr přeložen. Pokud se instalace zdařila, tak záměr přechází do stavu nainstalován. V tomto stavu je ONOS aplikaci, která daný záměr vytvořila, umožněno jeho kompletní odstranění například v případě, že již není potřeba nebo pokud není možné dosáhnout cíle, se kterým byl záměr vytvořen (Murrell, c2000–2019; Koshibe a Hart, 2016a).



Obrázek 8: Proces aplikace záměru na síť

Zdroj: zpracováno podle (Koshibe a Hart, 2016a)

Nejzajímavější vlastností subsystému *intent* framework je jeho schopnost překompilovat záměr v případě, že došlo k nějaké události v topologii. Pokud k ní dojde, je záměr překompilován tak, aby bylo opět dosaženo cíle. Díky této vlastnosti je umožněna automatizovaná správa sítě bez zásahu administrátora (Sanvito et al., 2018).

2.5 Aplikační a konfigurační subsystém v kontextu OSGi a Apache Karaf

Dříve než bude popsán aplikační a konfigurační subsystém, je zapotřebí popsat aplikační kontejner Apache Karaf a OSGi framework v kontextu ONOS kontroléru, resp. ONOS aplikací. Jádro kontroléru, jeho služby a ONOS aplikace jsou totiž napsány ve formě tzv. svazků[17]. Tyto svazky jsou načítány do aplikačního kontejneru Apache Karaf, který je postaven na vrcholu modulárního systému OSGi, a tvoří tzv. Karaf vlastnosti (Coker a Azodomolky, 2017, s. 189).

Technologie **OSGi** je sada specifikací, které definují dynamický systém komponent pro programovací jazyk Java. Jednotlivé komponenty, ze kterých se aplikace skládá, se označují jako svazky. Jeden svazek se skládá z jednoho Java ARchive (JAR) a manifestu. Manifest umožňuje identifikaci svazku v rámci OSGi a specifikaci jeho závislostí. Každá aplikace se skládá z alespoň jednoho svazku. Jednotlivé svazky mohou komunikovat lokálně nebo přes síť. OSGi umožňuje jejich dynamické spouštění v jednom Java Virtual Machine (JVM)[18], řízení jejich životního cyklu, vzájemnou komunikaci a poskytuje framework umožňující deklarovat některé jejich parametry jako konfigurovatelné. Framework umožňuje kromě změny a sledování hodnot konfigurovatelných parametrů za běhu i jejich načtení po restartu systému (Architecture – OSGi™ Alliance, c2019; Hohn, 2016; Vachuska a Higuchi, 2016).

OSGi však trpí jedním nedostatkem. Když je do OSGi kontejneru (u ONOS kontroléru Apache Karaf) instalován nový svazek, tak OSGi provádí kontrolu, zdali jsou dostupné všechny jeho závislosti a zdali neexistují konflikty mezi svazkem právě instalovaným a ostatními svazky. Nicméně tato kontrola neumožňuje žádné závislosti získat. OSGi pouze zkontroluje jejich dostupnost a zabrání nebo pozdrží instalaci svazku, pokud požadované závislosti nejsou dostupné. Je tedy nezbytné, aby sám uživatel OSGi identifikoval všechny svazky, které musí být v době instalace daného svazku v kontejneru již k dispozici. Nicméně to může být problém, neboť počet využívaných svazků může být v řádu desítek až stovek (Hohn, 2016).

Výše zmíněný problém řeší **aplikační kontejner Karaf**, pomocí konceptu tzv. vlastností. Vlastnost je skupina svazků, které by měly být instalovány společně. Je definována pomocí Extensible Markup Language (XML) souboru. V XML souboru se nachází například názvy svazků, ze kterých se vlastnost skládá (resp. způsob, jak s k nim dostat), a verze a jméno vlastnosti. Karaf také umožňuje specifikovat více vlastností v rámci jednoho XML souboru a vytvořit tak jejich repositář. K automatickému získávání závislostí využívá aplikační kontejner Karaf Maven[19] repositáře. Aplikační kontejner Karaf je možné ovládat lokálně i vzdáleně pomocí CLI. Skrze CLI je možné například řídit životní cyklus jednotlivých vlastností a zobrazit seznam nainstalovaných, resp. aktivovaných vlastností. ONOS CLI představuje pouze rozšíření CLI aplikačního kontejneru Karaf (Hohn, 2016; Koshibe a Prete, 2017).

---

[17] Správný překlad slova „bundle“ je balíček. Nicméně zde se nejedná o Java balíčky, z tohoto důvodu bylo slovo „bundle“ přeloženo jako svazek.

[18] JVM umožňuje spouštění programů napsaných v Javě na libovolném zařízení či operačním systému (Tyson, 2018).

[19] Maven je nástroj sloužící pro automatizací buildů aplikací (Robinson, 2013).

ONOS aplikace jsou definovány jako soubor Karaf vlastností. ONOS aplikace uchovává definice všech vlastností ze kterých se skládá (může se skládat i z jedné) v XML souboru a je zabalena do jednoho ONOS Application aRchive (OAR) souboru. OAR soubor obsahuje všechny artefakty, ze kterých se ONOS aplikace skládá (např. výše zmíněné definice vlastností a OSGi svazky) a představuje jednotku distribuce ONOS aplikace v rámci celého clusteru (Vachuska a Gaugusch, 2016).

2.5.1 Aplikační subsystém

Aplikační subsystém ONOS kontroléru je vybudován nad aplikačním kontejnerem Karaf. Účelem aplikačního subsystému je usnadnění správy ONOS aplikací v rámci celého ONOS clusteru. Ke správě ONOS aplikací v rámci jedné instance využívá aplikační subsystém aplikačního kontejneru Karaf a k replikaci svých ONOS aplikací v rámci ONOS clusteru eventuálně konzistentní mapu a *eastbound*/*westbound* API (Vachuska, 2015).

Eventuálně konzistentní mapa je založena na eventuálně konzistentním modelu, který se používá v distribuovaných systémech. Princip tohoto modelu spočívá v tom, že po určitém časovém intervalu, kdy nedošlo k aktualizaci, jsou replikované datové položky v rámci distribuovaného systému konzistentní. Nicméně u tohoto modelu se předpokládá, že než uplyne dostatečná doba bez aktualizace, může být datová položka v rámci distribuovaného systému nekonzistentní. Model eventuální konzistence je použitelný pouze pro aplikace odolné vůči chybám, tzv. *fault-tolerant* aplikace (Rouse a Wigmore, 2014).

2.5.2 Konfigurační subsystém

Konfigurační subsystém představuje nadstavbu nad frameworkem poskytovaným OSGi, který sice umožňuje měnit hodnoty konfigurovatelných parametrů svazku za běhu, ale pouze v rámci lokálního kontejneru. Lokálním kontejnerem je v případě ONOS kontroléru Apache Karaf určité instance. Konfigurační subsystém překrývá framework poskytovaný OSGi a provádí distribuci hodnot konfigurovatelných parametrů do celého clusteru. Díky konfiguračnímu subsystému je tedy zajištěno, že všechny instance budou mít stejnou konfiguraci (Vachuska a Higuchi, 2016).

Konfigurační subsystém v rámci jedné instance využívá frameworku poskytnutého OSGi ke sledování konfigurovatelných parametrů a k notifikaci jednotlivých svazků o jejich změnách. Navíc však umožňuje programátorovi umístit konfigurovatelné parametry do tzv. konfiguračního katalogu. Tím umožňuje správci sítě, který jeho ONOS aplikaci používá, zobrazení a nastavení hodnot konfigurovatelných parametrů pomocí ONOS CLI[20]. Konfigurační katalog také umožňuje konfiguračnímu subsystému zajistit, že změny v konfiguraci v rámci jedné instance budou propagovány do celého clusteru (Vachuska a Higuchi, 2016).

[20] ONOS GUI umožňuje pouze zobrazení konfigurovatelných parametrů s jejich současnými hodnotami bez možnosti jejich nastavení.

2.6 Flow rule subsystém

Podkapitola, pokud není uvedeno jinak, čerpá z části oficiální dokumentace ONOS kontroléru (Hart, 2016a).

Flow rule subsystém je zodpovědný za správu (aktualizace, odebírání a vkládání) všech *flow* pravidel v topologii. Pracuje s jednou autoritativní tabulkou, která je uložena v úložišti ONOS kontroléru. V této tabulce se nachází všechna *flow* pravidla, která se nachází, v dohledné době se budou nacházet nebo budou odebrána z *flow* tabulek SDN zařízení. *Flow rule* subsystém periodicky kontroluje, zdali záznamy v autoritativní tabulce korespondují se záznamy na jednotlivých SDN zařízeních. Při kontrole se také sbírají z jednotlivých záznamů statistiky (např. počet zpracovaných paketů) a je prováděna jejich aktualizace v autoritativní tabulce. Pokud je při kontrole objeven ve *flow* tabulce SDN zařízení záznam, který se nenachází v autoritativní tabulce, tak je odstraněn. Pokud je při kontrole zjištěn chybějící záznam ve *flow* tabulce SDN zařízení, který se nachází v autoritativní tabulce, tak je proveden pokus o jeho opětovné vložení. Kontrolou a prováděním změn ve *flow* tabulkách SDN zařízení pověřuje *flow rule* subsystém konkrétní implementaci *FlowRuleProvider*.

ONOS aplikace předávají požadavky na vkládání/odebírání *flow* pravidel *flow rule* subsystému pomocí *FlowRuleService* API ve formě objektů. Objekty reprezentující požadavky na vkládání *flow* pravidel lze vytvářet dvěma způsoby: buď lze vytvořit objekt typu *FlowRule* nebo *ForwardingObjective*. Pokud je vytvořen objekt typu *FlowRule*, tak je pro jeho předání *flow rule* subsystému přímo použito *FlowRuleService* API. Objekt tohoto typu je totiž možné ihned transformovat na konkrétní *flow* pravidlo. Objekt typu *ForwardingObjective* je naproti tomu transformován pomocí *FlowObjectiveService* API na kolekci *FlowRule* objektů, které jsou vytvořeny přímo na míru konkrétní konfiguraci *pipeline* určitého SDN zařízení. Teprve poté je tato kolekce objektů odeslána *flow rule* subsystému pomocí *FlowRuleService* API. Požadavky reprezentované pomocí objektů typu *ForwardingObjective* poskytují vyšší úroveň abstrakce. Umožňují programování datové vrstvy SDN zařízení nezávisle na konfiguraci jeho *pipeline*.

Flow rule subsystém také podporuje skládání elementárních operací, prováděných nad *flow* tabulkami SDN zařízení, do tzv. fází. Jednotlivé fáze jsou na sobě závislé. Nelze tedy začít provádět fázi, dokud ještě nebyla kompletně dokončena fáze předchozí. Díky tomu lze například předejít vzniku tzv. černé díry[21], která existuje do té doby, než je dokončena instalace cesty z jednoho koncového zařízení do druhého. Aby se předešlo jejímu vzniku, lze pomocí fází naplánovat vytvoření cesty tak, že bude vytvářena od jejího konce. Nemůže tím pádem dojít k tomu, že bude odeslán paket, který skončí v půli dosud nedokončené cesty v černé díře.

[21] Černá díra je místem v síti, kde jsou pakety zahozeny, aniž by byl informován odesilatel o neúspěšném doručení (Black Hole, c2019).

3 PRÁCE S ONOS KONTROLÉREM

Kapitola, pokud není uvedeno jinak, čerpá z části oficiální dokumentace ONOS kontroléru (Koshibe a Jianwei, 2018; Koshibe a Lantz, 2017).

V této kapitole bude uveden podrobný návod, jak nainstalovat, konfigurovat a ovládat ONOS kontrolér. Návod zahrnuje i konfiguraci operačního systému a je strukturován tak, aby mohl být použit jak k nasazení ONOS kontroléru do produkčního prostředí, tak i pro vývoj ONOS aplikací. Existují totiž dvě verze ONOS kontroléru: verze určená pro vývoj ONOS aplikací (zdrojový kód) neumožňující například spuštění ONOS kontroléru jako služby na pozadí a verze určená pro produkční prostředí neumožňující například ladící (*debug*) režim. K testování veškerých níže uvedených postupů byla použita verze ONOS kontroléru s názvem Quail[22] (2.0.0), která byla vydána 18. ledna 2019. Veškeré níže uvedené příkazy a postupy jsou validní na Long Term Support (LTS) distribuci Ubuntu (18.04). Části příkazů a Uniform Resource Locator (URL)[23], které uživatel musí doplnit, jsou označeny velkými písmeny.

3.1 Instalace

ONOS kontrolér je, jak již bylo uvedeno v kapitole 1, založený na Javě. Díky tomu je nezávislý na použitém operačním systému – funguje na všech platformách pro které je dostupné Java Runtime Environment (JRE)[24], které je součástí Java Development Kit (JDK)[25]. Lze ho však nainstalovat i samostatně. Nicméně se i pro pouhé spuštění (pro vývoj ONOS aplikací je to nezbytné) doporučuje nainstalovat kompletní JDK[26], neboť aplikační kontejner Karaf vyžaduje nastavení proměnné prostředí JAVA_HOME. Tou lze sice odkazovat přímo na nainstalované JRE, nicméně v takovém případě, na rozdíl od jiných Java programů, nelze ONOS kontrolér vůbec spustit. Aplikační kontejner Karaf totiž v takovém případě považuje cestu k proměnné za neplatnou. Kromě JDK vyžaduje verze určená pro produkční prostředí také instalaci nástroje curl[27]. Verze určená pro vývoj ONOS aplikací a samotného kontroléru (zdrojový kód) vyžaduje navíc instalaci webového prohlížeče Google Chrome, verzovacího systému Git a nástroje Bazel[28] umožňujícího build samotného zdrojového kódu. K testovacím a studijním účelům se doporučuje také nainstalovat síťový emulátor Mininet, který dokáže simulovat libovolnou

[22] Quail znamená česky křepelka. Všechny verze ONOS kontroléru nesou anglická jména ptáků.
[23] URL je adresa zdroje v internetu. Obsahuje jak jeho umístění, tak i protokol, který se používá k přístupu (Uniform Resource Locator, c2019).
[24] JRE je sada softwarových nástrojů umožňující běh Java aplikací na libovolné platformě. Skládá se z: JVM, základních tříd a podpůrných knihoven (Java Runtime Environment (JRE), c2019).
[25] JDK je soubor nástrojů umožňující vývoj Java aplikací. Osahuje: JRE, interpret a kompilér Java kódu, generátor dokumentace a další (Differences between JDK, JRE and JVM, 2015).
[26] V oficiální dokumentaci je uvedená jako doporučená verze Oracle Java 1.8.
[27] Curl je nástroj umožňující pomocí CLI či skriptů přenášet data pomocí různých protokolů aplikační vrstvy (Curl, 1997).
[28] Bazel je nástroj umožňující buildování a testování aplikací vyvíjených například v Javě, C++, Go a mnoha dalších programovacích jazycích. Funguje na operačních systémech Windows, MacOS a Linux. Díky lokálnímu a distribuovanému cachování, optimalizované analýze závislostí a paralelismu umožňuje rychlé, efektivní a inkrementální buildování aplikací (Bazel, c2019).

topologii a umožňuje ONOS kontroléru spojení s SDN zařízeními, ze kterých se simulovaná topologie skládá.

Hardwarové požadavky ONOS kontroléru se těžko definují, neboť závisí na mnoha faktorech. Například na velikosti clusteru a spravované topologie, počtu aktivovaných ONOS aplikací, počtu zpráv vyměňovaných mezi kontrolérem a síťovými zařízeními atd. V oficiální dokumentaci jsou uváděny jako minimální následující požadavky: *central processing unit* (CPU) s 2 jádry, 2 GiB Random Access Memory (RAM) a 10 GiB volného místa na pevném disku. Nicméně například pro build ONOS kontroléru pomocí nástroje Bazel nemusí stačit ani 4 GiB RAM. Může totiž docházet k chybě týkající se nedostatku RAM a tím pádem i k selhání celého buildu. Jako optimální požadavky na hardware umožňující i hladký vývoj ONOS aplikací byly na základě praktických zkušeností stanoveny následující: CPU se 4 jádry, 6 GiB RAM a 30 GiB volného místa na pevném disku.

Kromě požadavků na hardware a software má ONOS kontrolér také jisté požadavky na konektivitu. Podle oficiální dokumentace je sice již nyní možné ONOS kontrolér nainstalovat a spustit bez připojení do internetu, nicméně je stále vyžadováno při stahování ONOS balíčků při jeho buildu a samozřejmě také ke stažení softwaru třetích stran, který je vyžadován jako prerekvizita (vizte první odstavec). ONOS kontrolér také ve výchozím nastavení[29] vyžaduje, aby byly otevřeny následující porty: 8181 – REST API a GUI, 8101 – přístup k CLI, 9876 – komunikace mezi instancemi clusteru, 6653 – komunikace s SDN zařízeními pomocí protokolu OpenFlow a 6640 – komunikace s SDN zařízeními pomocí protokolu OVSDB.

Níže je uveden návod k instalaci, resp. buildu ONOS kontroléru. Návod je rozdělen do čtyř částí: nejprve budou uvedeny společné prerekvizity, dále síťový emulátor Mininet a na závěr zvlášť postup, jak nainstalovat obě verze ONOS kontroléru (verze pro produkční prostředí a verze určené pro vývoj). Na oficiálních stránkách[30] lze také stáhnout předpřipravený obraz určený pro multiplatformní virtualizační nástroj Oracle VirtualBox. Tento obraz obsahuje nainstalovaný a plně funkční ONOS kontrolér (Peacock 1.15) a Mininet. Umožňuje tak přeskočit části návodu popisující instalaci společných prerekvizit, síťového emulátoru Mininet a produkční verze ONOS kontroléru.

[29] Změna konfigurace přístupových portů (soubory lze nalézt v kořenovém adresáři ONOS kontroléru): GUI a REST API – soubor *org.ops4j.pax.web.cfg*, CLI - *org.apache.karaf.shell.cfg*, OpenFlow a OVSDB – představují konfigurovatelné parametry zásuvných modulů umožňujících podporu konkrétních *southbound* protokolů (vizte druhý odstavec kapitola 3.4).
[30] https://drive.google.com/file/d/1JcGUJJDTtbHNnbFzC7SUK52RmMDBVUry/view

Společné prerekvizity

Příkaz pro instalaci Oracle JDK 1.8 v jenom kroku (bez nastavení JAVA_HOME) – použití jiného postupu může později vést k selhání buildu ONOS kontroléru:

```
sudo apt-get install software-properties-common -y && \
sudo add-apt-repository ppa:webupd8team/java -y && \
sudo apt-get update && \
echo "oracle-java8-installer shared/accepted-oracle-license-v1-1 select true" |
sudo debconf-set-selections && \
sudo apt-get install oracle-java8-installer oracle-java8-set-default -y
```

Příkaz umožňující zjištění cesty, kam bylo nainstalováno Oracle JDK:

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

Příkaz sloužící k otevření textového souboru obsahujícího proměnné prostředí pro všechny procesy */etc/environment* v textovém editoru nano:

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

Následující řádek je zapotřebí přidat do textového souboru */etc/environment* a poté tento soubor uložit (hodnota v uvozovkách představuje cestu zjištěnou ve druhém příkazu):

```
JAVA_HOME="/usr/lib/jvm/java-8-oracle"
```

Příkaz sloužící k aplikaci proměnné JAVA_HOME:

```
source /etc/environment
```

Příkaz sloužící k instalaci nástroje curl:

```
sudo apt install curl
```

Ke stažení zdrojového kódu síťového emulátoru Mininet a ONOS kontroléru je vyžadován verzovací systém Git. Následující příkaz slouží k jeho instalaci:

```
sudo apt-get install git
```

Mininet

Ke spuštění síťového emulátoru Mininet je vyžadována utilita *ifconfig*, která je obsažena v balíčku síťových nástrojů *net-tools*. Následující příkaz slouží k jeho instalaci:

```
sudo apt install net-tools
```

Příkazy sloužící ke stažení zdrojového kódu síťového emulátoru Mininet do aktuálního adresáře:

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

Příkaz sloužící k přepnutí do adresáře síťového emulátoru Mininet:

```
cd mininet
```

Příkaz sloužící k vypsaní seznamu dostupných verzí síťového emulátoru Mininet:

```
git tag
```

Příkaz sloužící k přepnutí na konkrétní verzi síťového emulátoru Mininet (doporučuje se nejnovější verze 2.2.2):

```
git checkout -b MININET_VERZE
```

Příkaz sloužící ke kompletní instalaci síťového emulátoru Mininet, kromě POX kontroléru[31]:

```
mininet/util/install.sh –nfv
```

Verze určená pro produkční prostředí

Verzi pro produkční prostředí se doporučuje instalovat do adresáře */opt*, který se nachází v kořenovém adresáři. Pokud tento adresář neexistuje, následující příkaz umožňuje jeho vytvoření – vzhledem k tomu, že se nejedná o adresář uživatele, je nutné tento a všechny ostatní příkazy spouštět pod super uživatelem (*sudo*):

```
sudo mkdir /opt
```

Příkaz sloužící k přesunu do adresáře */opt*:

```
cd /opt
```

Příkaz sloužící ke stažení zkomprimovaného souboru obsahujícího požadovanou verzi ONOS kontroléru do aktuálního adresáře (konkrétní verzi lze vybrat z oficiálních stránek[32]):

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

Příkaz sloužící k rozbalení zkomprimovaného souboru:

```
sudo tar xzf onos-ONOS_VERZE.tar.gz
```

Příkaz sloužící k přejmenování rozbaleného adresáře, který obsahuje ONOS kontrolér:

```
sudo mv onos-ONOS_VERZE onos
```

Verze určená pro vývoj

Příkaz sloužící k instalaci prerekvizit pro buildovací nástroj Bazel:

```
sudo apt-get install pkg-config zip g++ zlib1g-dev unzip python
```

Příkaz sloužící ke stažení zdrojového kódu buildovacího nástroje Bazel do aktuálního adresáře (konkrétní verzi lze vybrat z repositáře Github[33]):

```
wget https://github.com/bazelbuild/bazel/releases/download/VERZE/bazel-VERZE-
installer-linux-x86_64.sh
```

Příkaz sloužící ke změně práv instalačního skriptu (umožňuje jeho spuštění) buildovacího nástroje Bazel:

```
chmod +x bazel-VERZE-installer-linux-x86_64.sh
```

---

[31] Je možné, že na novějších verzích operačního systému Linux (např. Ubuntu 18.04) nastane problém s tím, že Mininet nepůjde nainstalovat z důvodu nedostupnosti starého balíčku *iproute*. V takovém případě je nutné přepsat v instalačním skriptu (*install.sh*) *iproute* na *iproute2*.

[32] https://wiki.onosproject.org/display/ONOS/Downloads

[33] https://github.com/bazelbuild/bazel/releases

Příkaz sloužící k instalaci buildovacího nástroje Bazel do domovského adresáře aktuálního uživatele:

```
./bazel-VERZE-installer-linux-x86_64.sh --user
```

K testování některých částí ONOS kontroléru při jeho buildu je vyžadován webový prohlížeč Google Chrome. Následující příkazy slouží k jeho instalaci:

```
sudo apt install gdebi-core
wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
sudo gdebi google-chrome-stable_current_amd64.deb
```

Příkaz sloužící ke stažení zdrojového kódu ONOS kontroléru do aktuálního adresáře:

```
git clone https://gerrit.onosproject.org/onos
```

Utility skripty jsou umístěny v různých podadresářích kořenového adresáře ONOS kontroléru. V oficiální dokumentaci je z tohoto důvodu doporučeno nastavit cestu k jeho kořenovému adresáři. Díky tomu je poté možné používat různé utility skripty z libovolného aktuálního adresáře. Nicméně na základě praktických zkušeností s ONOS kontrolérem se ukázalo, že je to nezbytné, neboť bez toho není možné provést jeho build (vždy na stejném místě dochází k chybě). K nastavení je zapotřebí otevřít a upravit skript */home/uziv_jmeno/.profile*, který se spouští při každém přihlášení uživatele. Je zapotřebí do něj přidat dva příkazy, které specifikují cestu ke kořenovému adresáři ONOS kontroléru. Následující příkaz slouží k otevření souboru *.profile* v textovém editoru nano:

```
nano /home/UZIVATEL/.profile
```

Do souboru je zapotřebí přidat dva následující řádky (předpokládá se, že byl zdrojový kód umístěn do domovského adresáře uživatele), následně provést jeho uložení a poté systém restartovat:

```
export ONOS_ROOT=~/onos
source $ONOS_ROOT/tools/dev/bash_profile
```

První build ONOS kontroléru může trvat jednotky až desítky minut v závislosti na hardwaru. Před vlastním buildem ONOS kontroléru by se měl uživatel ujistit, zdali jsou volné alespoň 2 GiB RAM. Jinak může docházet k selhání buildu. Následující příkaz slouží k buildu ONOS kontroléru pomocí nástroje Bazel – uživatel se před exekucí tohoto příkazu musí přemístit do kořenového adresáře ONOS kontroléru:

```
bazel build onos
```

3.2 Spouštění

V této podkapitole bude popsáno spouštění ONOS kontroléru. Podkapitola bude rozdělena na dvě části: spouštění verze určené pro produkční prostředí a verze určené pro vývoj. Součástí spouštění verze pro produkční prostředí bude také návod, jak spustit ONOS kontrolér jako službu.

Spouštění verze pro produkční prostředí
Použitím následujícího příkazu dojde ke spuštění ONOS kontroléru a zároveň se v terminálu, ve kterém byl tento příkaz spuštěn, otevře jeho CLI:

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

Příkazy umístěné pod tímto odstavcem se týkají spuštění ONOS kontroléru jako služby. Pokud je ONOS kontrolér spuštěn jako služba, tak je automaticky spouštěn při startu operačního systému a restartován, pokud se vyskytne chyba, která způsobí jeho ukončení. Konfigurace spuštění procesu jako služby se u operačního systému liší podle použitého systému správy služeb (*systemd*, *upstart, sysV*). Zde bude popsána konfigurace pro systém správy služeb *systemd*. Na oficiálních stránkách[34] lze také nalézt konfiguraci pro systém správy služeb *upstart*. Následující příkaz slouží k přidání uživatele *sdn*, pod kterým bude ONOS kontrolér jako služba běžet:

```
sudo adduser sdn --system --group
```

Následující příkazy slouží k instalaci souborů umožňujících spuštění ONOS kontroléru jako služby pro *systemd*:

```
sudo cp /opt/onos/init/onos.initd /etc/init.d/onos
sudo cp /opt/onos/init/onos.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable onos
```

ONOS kontrolér, pokud je spuštěn jako služba, načítá konfiguraci ze souboru */opt/onos/options*. Do tohoto souboru je zapotřebí přidat uživatele, pod kterým bude ONOS kontrolér běžet. Následující příkaz slouží k otevření konfiguračního souboru (pokud soubor neexistuje, tak je při uložení vytvořen):

```
sudo nano /opt/onos/options
```

Následující řádek je zapotřebí přidat do souboru, aby ONOS běžel jako služba pod uživatelem *sdn* (vytvořen pomocí prvního příkazu této sekce):

```
ONOS_USER=sdn
```

Příkaz sloužící ke spuštění, zastavení, restartování a kontrole stavu služby (z možností ve složených závorkách je nutné zvolit pouze jednu):

```
sudo systemctl {start|stop|status|restart} onos.service
```

[34] https://wiki.onosproject.org/display/ONOS/Running+ONOS+as+a+service

Spuštění verze určené pro vývoj
Příkaz pro spuštění verze ONOS kontroléru určené pro vývoj, který zároveň provede i jeho build, pokud ještě nebyl proveden – nutno provádět ve složce se zdrojovým kódem (v terminálu, ze kterého byl tento příkaz spuštěn, se bude až do ukončení ONOS kontroléru zobrazovat jeho log):

```
bazel run onos-local
```

Nepovinné parametry:

- `clean` slouží k čisté lokální instalaci (ztráta konfigurace).
- `debug` umožňuje vzdálené ladění kódu.

3.3 Rozhraní umožňující interakci s ONOS kontrolérem

ONOS umožňuje uživateli interakci skrze tři rozhraní: REST API, CLI a GUI. Ke každému z nich se přistupuje jiným způsobem a každé nabízí jiné možnosti interakce a různý uživatelský komfort. V této podkapitole bude popsáno: jak k jednotlivým rozhraním přistupovat, jaké nabízejí uživateli možnosti interakce a jak konfigurovat zabezpečení přístupu.

REST API slouží primárně k interakci s ONOS kontrolérem pomocí externích webových aplikací nebo utilit. Pro účely testování a ladění je možné využít i přímého přístupu například pomocí nástroje curl či Postman. REST API se nedoporučují využívat pro směrování a k programování vysoce výkonných síťových aplikací, neboť by z důvodu distribuované architektury rozhraní docházelo ke zvyšování odezvy. Jádrem poskytovaná REST API umožňují například: management ONOS aplikací a zásuvných modulů umožňujících podporu *southbound* protokolů, síťovou správu založenou na síťových politikách, management *flow* pravidel na SDN zařízeních, sledování sítě a další. Další REST API mohou poskytovat vlastní ONOS aplikace. K veškerým dostupným REST API poskytovaným jádrem ONOS kontroléru lze přistupovat pomocí následující URL (výchozí port – 8181):

```
http://IP_ADRESA_ONOS_KONTROLÉRU:PORT/onos/v1/KONKRETNI_REST_API
```

Automaticky generovanou dokumentaci obsahující kompletní výčet jádrem poskytovaných REST API (včetně jednotlivých operací a účelu) lze nalézt na následující URL:

```
http://IP_ADRESA_ONOS_KONTROLERU:PORT/onos/v1/docs/
```

ONOS CLI je hlavním administrativním rozhraním ONOS kontroléru umožňující jeho ovládání pomocí široké palety příkazů. Dostupné příkazy umožňují ekvivalentní možnosti interakce jako REST API poskytována jádrem ONOS kontroléru. Kompletní výpis obsahující všechny příkazy lze vyvolat stisknutím klávesy „TAB“ v ONOS CLI. (Jeho stručnou a neaktuální verzi lze také nalézt na oficiálních stránkách.[35]) Stejně jako REST API je možné ONOS CLI rozšířit o další příkazy, které je možné implementovat v rámci vlastních ONOS aplikací. Následující příkaz umožňuje lokální/vzdálený zabezpečený přístup k ONOS CLI (výchozí port – 8101):

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

```
ssh -p PORT UZIVATELSKE_JMENO@IP_ADRESA_ONOS_INSTANCE
```

GUI ONOS kontroléru představuje tzv. *single page application*[36] a nabízí ze všech poskytovaných rozhraní nejvyšší uživatelský komfort. Nicméně oproti REST API a CLI nabízí daleko méně možností interakce. Například neumožňuje správu sítě založenou na síťových politikách, management *flow* pravidel atd. Jeho primárním účelem je vizualizace topologie[37], sledování stavu sítě a management životního cyklu ONOS aplikací. Stejně jako REST API a CLI je možné ho rozšířit. Rozšíření se provádí injektováním pohledů vlastních ONOS aplikací. Pro přístup ke GUI ONOS kontroléru pomocí webového prohlížeče slouží následující URL (výchozí port – 8181):

```
http://IP_ADRESA_ONOS_KONTROLERU:PORT/onos/ui/index.html
```

3.3.1 Zabezpečení přístupu

Ve výchozím nastavení jsou k dispozici dva uživatelské účty umožňující přístup do libovolného rozhraní ONOS kontroléru – *onos/rocks* a *karaf/karaf*. Jejich definice se nacházejí v souboru *users.properties*, jehož umístění se liší podle toho, zdali se jedná o verzi určenou pro produkční prostředí nebo pro vývoj. Úpravou tohoto souboru je možné editovat/přidat/smazat uživatele a vytvořit/editovat/smazat skupinu, do které je uživatel přiřazen. Práva pro skupinu určují role, které jsou skupině přiřazeny. K dispozici jsou dva druhy rolí (práva přidělena na základě povolených příkazů): *admin* – umožněna kompletní správa ONOS kontroléru (instalace nových ONOS aplikací atd.) a *viewer* – umožněno pouze sledování stavu sítě a aktivace či deaktivace ONOS aplikací. Problémem celého zabezpečení je fakt, že uživatelská hesla jsou v souboru *users.properties* uložená v nezašifrované podobě (*plain text*). Je tedy zapotřebí k tomuto souboru zamezit přístup, a to i včetně jeho čtení (Vachuska a Koshibe, 2016).

Kromě úpravy výše zmíněného souboru je možné spravovat přístupy také pomocí utility skriptu *onos-user-password* a skrze ONOS CLI, resp. aplikační kontejner Karaf. Kompletní návod, jak pracovat s uživatelskými účty pomocí ONOS CLI je dostupný na oficiálních stránkách aplikačního kontejneru Karaf[38]. Následující příkaz umožňuje správu přístupů pomocí utility skriptu *onos-user-password* (uživatel se musí nacházet v adresáři, ve kterém je utilita uložena a musí zvolit jednu z možností ve složených závorkách):

```
./onos-user-password UZIVATEL {HESLO|--remove}
```

- Nově vytvořený uživatel pomocí tohoto příkazu je automaticky zařazen do skupiny *admingroup*, které jsou přiřazeny role *admin* a *viewer*.

---

[36] *Single page application* (SPA) je typ webové aplikace, která dynamicky načítá pouze vybrané prvky stránky, které souvisí s interakcí uživatele. Odpovědí na požadavek uživatele není kompletní webová stránka, ale pouze dokument ve formátu JSON. Aktualizace vybraných prvků jsou prováděny až na klientovi na základě získaného dokumentu. Tím je zabráněno kompletnímu načítání nových stránek ze serveru a čekání na vizuální interpretaci stránky webovým prohlížečem. Díky tomu se SPA aplikace jeví uživateli jako mnohem rychlejší (Ismail, 2018).
[37] Správné zobrazení topologie je limitováno počtem zařízení. Již při dvaceti zařízeních ztrácí zobrazení vypovídající hodnotu.
[38] https://karaf.apache.org/manual/latest/security

3.4 Management aplikací

Společně s ONOS kontrolérem jsou dodávány ONOS aplikace, které implementují různé síťové funkcionality (např. směrování nebo přepínání), a zásuvné moduly umožňující podporu různých *southbound* protokolů. (Ve zbytku této podkapitoly označovány souhrnně jako aplikace.) Aplikace se mohou nacházet ve dvou stavech: nainstalované a aktivované. Nainstalované aplikace se nacházejí v adresáři */apps*, který se nachází v kořenovém adresáři ONOS kontroléru, a jsou připraveny k aktivaci. Aktivované aplikace jsou spouštěny při každém startu ONOS kontroléru. Uživatel by měl mít aktivovány pouze aplikace, které skutečně potřebuje. Počet aktivovaných aplikací může ovlivňovat dobu zpracování paketu[39] na ONOS kontroléru (paket je ke zpracování předáván aplikacím postupně) a jeho hardwarové nároky. Počet aktivovaných aplikací ve výchozím nastavení se liší podle toho, zdali se jedná o verzi určenou pro vývoj nebo pro produkční prostředí. V tabulce 2 jsou uvedeny aplikace, které musí být aktivovány, aby fungovalo alespoň jednoduché směrování[40] paketů v SDN sítích, které využívají protokol OpenFlow. Kromě řízení životního cyklu, umožňuje ONOS kontrolér také konfiguraci parametrů aplikace za běhu (např. priorita *flow* pravidel vložených danou aplikací). Nicméně konfigurace parametrů se neukládá a je potřeba ji provádět znovu po každém startu ONOS kontroléru.

Tabulka 2: Aplikace umožňující jednoduché směrování paketů

Aplikace	**Název***
Default drivers#+	org.onosproject.drivers
Host Location Provider+	org.onosproject.hostprovider
LLDP Link Provider+	org.onosproject.lldpprovider
OpenFlow Base Provider+	org.onosproject.openflow-base
OpenFlow Provider Suite+	org.onosproject.openflow
Host Mobility+	org.onosproject.mobility
Reactive Forwarding+	org.onosproject.fwd

* Pod tímto názvem je možné aplikaci dohledat v CLI ONOS kontroléru, resp. v jeho souborovém systému. Název v reverzní DNS notaci se také používá ke správě jednotlivých aplikací.
# Aplikace aktivována ve výchozím nastavení na verzi pro produkční prostředí.
+ Aplikace aktivována ve výchozím nastavení na verzi pro vývoj.

Zdroj: vlastní zpracování

Jak již bylo uvedeno v kapitole 3.3, ONOS kontrolér umožňuje management aplikací pomocí veškerých dostupných přístupových rozhraní. V GUI ONOS kontroléru lze provádět management životního cyklu a sledování hodnot konfigurovatelných parametrů aplikací skrze záložky „Applications“ a „Settings“.[41] REST API umožňuje navíc kromě sledování hodnot konfigurovatelných parametrů také jejich konfiguraci. Nicméně management aplikací pomocí REST API je z pohledu uživatele velmi nepraktický, neboť utility skript umožňující alespoň

[39] Jednotky přenosu informace z libovolné ISO/OSI vrstvy určené ke zpracování jsou ONOS kontroléru vždy zasílány ve formě paketů.
[40] Vizte kapitola 4.3 – Reactive Forwarding
[41] Obě záložky je možné najít v *dropdown* navigačním menu ONOS kontroléru.

pohodlný management životního cyklu, poskytuje pouze verze určená pro vývoj.[42] Z tohoto důvodu bude níže podrobně popsán pouze management aplikací skrze ONOS CLI, které poskytuje stejné možnosti jako REST API. Všechny následující příkazy jsou zadávány do ONOS CLI (připojení vizte kapitola 3.3).

Příkaz sloužící k výpisu veškerých nainstalovaných aplikací (aktivované jsou označeny hvězdičkou), včetně podrobností (název, verze, práva atd.):

```
apps
```

Nepovinné parametry:

- `-s` slouží ke zobrazení zkráceného výpisu, který obsahuje pouze název v reverzní DNS notaci, verzi a jméno aplikace.
- `-a` umožňuje vypsat pouze aktivované aplikace.

Příkaz umožňující správu životního cyklu aplikace (z každé složené závorky je zapotřebí vybrat jednu z uvedených možností):

```
app {install|activate|deactivate|uninstall} {NAZEV|URL}
```

- V případě, že chce uživatel aktivovat (*install*), deaktivovat (*deactivate*) nebo odinstalovat (*uninstall*) aplikaci použije její název v reverzní DNS notaci, který je možné zjistit pomocí prvního uvedeného příkazu.
- URL je zapotřebí zadávat pouze v případě, pokud chce uživatel aplikaci nainstalovat (*install*). Pokud uživatel instaluje aplikaci ve formátu OAR z lokálního pevného disku, musí být URL ve tvaru *file:///CESTA/NAZEV_SOUBORU.oar*.

Příkaz umožňující nastavení hodnot konfigurovatelných parametrů aplikace (název aplikace je nutné zadávat ve formátu reverzní DNS notace):

```
cfg set NAZEV_APLIKACE NAZEV_PARAMETRU HODNOTA
```

Příkaz sloužící k výpisu konfigurovatelných parametrů aplikací (parametry v hranatých závorkách jsou nepovinné, při kombinaci je však nutné dodržet jejich pořadí):

```
cfg [-s] [get] [NAZEV_APLIKACE] [NAZEV_PARAMETRU]
```

Nepovinné parametry:

- `-s` slouží ke zobrazení zkráceného výpisu ve formátu *název parametru=hodnota*.
- `get` slouží k vypsaní hodnoty určitého konfigurovatelného parametru nebo hodnot veškerých konfigurovatelných parametrů dané aplikace.

[42] Konkrétně se jedná o utility skript *onos-app*, který poskytuje funkcionality usnadňující testování při vývoji vlastních ONOS aplikací (modulů). Jeho použití bude popsáno v kapitole 5.2.

4 FUNKCIONALITY ONOS KONTROLÉRU

V této kapitole budou popsány funkcionality, které jsou dodávány společně s ONOS kontrolérem nebo takové u nichž je ONOS kontrolér součástí jejich architektury. Funkcionality dodávané společně s ONOS kontrolérem musí být nejprve schváleny komunitou. V této kapitole tedy nebudou brány v potaz funkcionality vyvíjené nezávislými vývojáři. Takové funkcionality je možné nalézt nejčastěji v repositářích webové služby GitHub a dodatečně je do kontroléru doinstalovat. Jedná se většinou o jednoduché ONOS aplikace sloužící zejména pro začínající ONOS programátory ke studijním účelům. Funkcionality dodávané společně s ONOS kontrolérem jsou poskytovány ve formě ONOS aplikací a zásuvných modulů umožňujících podporu *southbound* protokolů.

4.1 Hlavní funkcionality určené pro poskytovatele služeb

Tato kapitola bude věnována hlavním pokročilým funkcionalitám ONOS kontroléru, které jsou určeny zejména pro poskytovatele služeb. Funkcionality, které budou v následujících odstavcích popisovány jsou uvedeny v abecedním pořadí v tabulce 3.

Tabulka 3: Pokročilé funkcionality určené pro poskytovatele služeb

Funkcionalita	**Název***
Castor	org.onosproject.castor
Multicast Forwarding	org.onosproject.mfwd, org.onosproject.pim
Packet Optical Convergence	org.onosproject.optical-rest, org.onosproject.newoptical, org.onosproject.optical-model, org.onosproject.drivers.optical
Proxy ARP/NDP	org.onosproject.proxyarp
SDN-IP	org.onosproject.sdnip
SDN-IP Reactive routing	org.onosproject.reactive-routing
Virtual Broadband Network Gateway	org.onosproject.virtualbng

* Pod tímto názvem/y je možné funkcionalitu, resp. její část dohledat v CLI ONOS kontroléru, resp. v jeho souborovém systému. Název v reverzní DNS notaci se také používá ke správě jednotlivých funkcionalit. Ke zprovoznění některých funkcionalit je zapotřebí nainstalovat více ONOS aplikací a zásuvných modulů umožňujících podporu *southbound* protokolů.

Zdroj: vlastní zpracování

Castor umožňuje veřejný *peering*[43] mezi autonomními systémy (AS) pomocí *peeringových* uzlů[44] a privátní *peering* mezi společnostmi a poskytovateli cloudových služeb. Veřejný *peering* pomocí Castor modulu má oproti tradičnímu modelu následující výhody: lepší způsob uplatňování IXP politik, lepší management ARP, dotazy mohou být poslány přímo příslušnému *peeru*, a vylepšenou telemetrii. Privátní *peering* oproti tradičnímu modelu má následující výhody: lepší škálovatelnost, možnost nastavení elastické šířky pásma mezi *peery* a umožňuje

[43] *Peering* označuje přímé propojení počítačových sítí telekomunikačních společností za účelem výměny datového provozu (What is peering?, c2019).
[44] Internet Exchange Point (IXP) je fyzický přístupový bod, skrze který poskytovatelé služeb propojují své sítě a vyměňují si svá data. IXP byl vytvořen z toho důvodu, aby data mezi sousedními sítěmi neputovala přes drahé telekomunikační sítě třetí stran a aby bylo dosaženo vyšší rychlosti (Internet Exchange Point (IXP), c2019).

zákazníkům nastavovat a spouštět aplikace a služby v cloudovém prostředí bez přímého zásahu poskytovatele internetového připojení nebo poskytovatele služeb (Kumar, 2017; Rouse, 2011b).

Multicast Forwarding slouží k přeposílání paketů v IP protokolu z jednoho zdroje skupině, která se skládá z více koncových zařízení. (Jeho architektura se skládá ze čtyř primárních funkcionalit: směrovací tabulky, sloužící ke správě směrovacích stavů, modulu Multicast Forwarding, který reaguje na *multicast* komunikaci, manažera *multicast* záměrů zodpovědného za vytvoření cest skrze síť a CLI a REST API umožňující správci a externím aplikacím modifikaci existujících směrovacích stavů.) K efektivnímu směrování *multicast* komunikace využívá modul Multicast Forwarding Protocol-Independent Multicast (PIM). Jedná se o protokol sloužící pro efektivní směrování *multicast* komunikace mezi doménami. Tento protokol není závislý na konkrétním použitém *unicast* směrovacím protokolu (Eddy, 2015; Estrin et al., 1997).

Packet Optical Convergence slouží ke správě sítě skládající se ze zařízení využívající k přenosu dat dvě různé technologie: přenos dat prostřednictvím světelných signálů a elektromagnetický přenos. Packet Optical Convergence poskytuje konvergovaný pohled na topologii, který umožňuje rychlé zavádění nových služeb a programovatelnost sítě jako celku pomocí *intent* frameworku (Koshibe, 2016).

Proxy ARP/NDP umožňuje směrovači odpovídat na ARP nebo Neighbor Discovery Protocol (NDP) dotazy, které jsou ve skutečnosti určeny pro jiný stroj. Směrovač zasílá svou fyzickou MAC adresu jako odpověď a tazatel poté komunikuje se směrovačem, který má za úkol přeposlat jeho dotazy skutečnému adresátovi. Hlavní výhodou proxy ARP/NDP je, že lze použít jediný směrovač pro komunikaci se všemi koncovými zařízeními v síti. Nevýhoda proxy ARP/NDP spočívá v tom, že koncové zařízení si myslí, že je možné všechna ostatní zařízení dosáhnout pomocí ARP/NDP dotazu. Kvůli tomu dochází ke zvětšování ARP/NDP tabulek (Carl-Mitchell a Quarterman, 1987; Hart, 2016b).

SDN-IP umožňuje připojit SDN síť k externím sítím (internetu) pomocí Border Gateway Protocol (BGP) protokolu. Externě z pohledu BGP se SDN síť jeví jako jeden AS, který funguje jako libovolný tradiční AS. V rámci AS poskytuje SDN-IP integrační mechanismy umožňující komunikaci mezi ONOS kontrolérem a BGP protokolem. Na úrovni protokolu BGP se SDN-IP chová jako klasický BGP *speaker*. Termín BGP *speaker* označuje v tradiční síťové architektuře směrovač, který využívá ke směrování BGP protokol. V SDN síti se může nacházet jedna nebo více spuštěných instancí SDN-IP modulu, přičemž každá představuje jeden BGP *speaker*. Instance používají *external* BGP (eBGP) k výměně BGP směrovacích informací s hraničními směrovači sousedních externích sítí a *internal* BGP (iBGP) k šíření těchto informací mezi sebou. Z pohledu ONOS kontroléru je SDN-IP modulem, který používá služby kontroléru ONOS k programování datové vrstvy SDN zařízení. Typicky se používá SDN síť řízená

SDN-IP modulem jako tranzitní[45] AS, který spojuje různé IP sítě. SDN-IP zajišťuje pouze proaktivní[46] směrování tranzitní komunikace, o zbytek se stará další modul, který je na SDN-IP navázán, jmenovitě SDN-IP Reactive Routing (Koshibe a Hart, 2016b).

SDN-IP Reactive Routing je závislý na SDN-IP. Plní funkci virtuální výchozí brány a stará se o reaktivní[47] směrování komunikace mezi koncovými zařízeními v případě, že se alespoň jedno z nich nachází uvnitř SDN sítě (Lin, 2016).

Virtual Broadband Network Gateway umožňuje připojení do internetu koncovým zařízením s privátní IP adresou. Poskytuje tři základní funkcionality:

- Zpracovává REST požadavky od koncových zařízení na přidělení veřejné IP adresy. V případě úspěchu je odpovědí na požadavek přidělená veřejná IP adresa.
- Udržuje mapování privátních IP adres na veřejné IP adresy.
- Vytváří přímá spojení pro zařízení s privátní IP adresou, která umožní daným zařízením přístup do internetu (Lin a Hart, 2015).

4.2 Central Office Re-architected as a Datacenter (CORD)

Podkapitola, pokud není uvedeno jinak, čerpá z oficiálních stránek projektu CORD (CORD Platform, c2019).

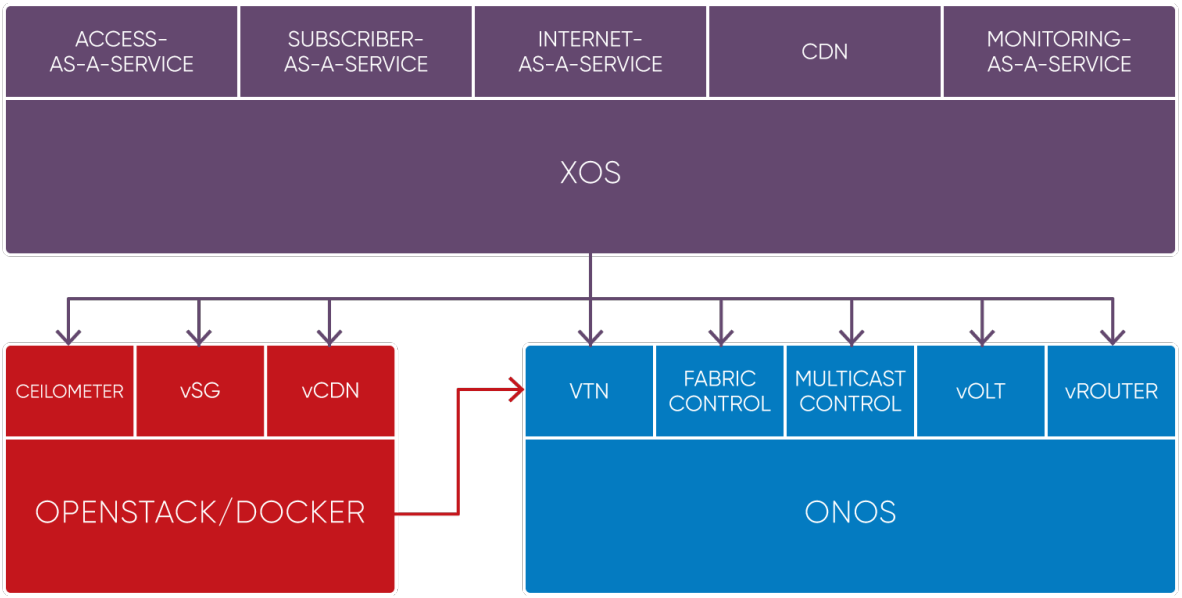
Hranice sítě operátorů[48] (například *head-end*[49] u poskytovatelů kabelové televize) je místem, kde se operátoři připojují ke svým zákazníkům. Záměrem projektu CORD je transformovat tuto hranici na agilní platformu pro poskytování služeb. Díky tomu mohou operátoři nabízet svým zákazníkům služby v té nejvyšší kvalitě spolu s inovativními službami nové generace. CORD byl původně součástí projektu ONOS, později z něj však byl vyňat, a proto mu je věnována zvláštní podkapitola. ONOS kontrolér je pouze součástí architektury platformy CORD (vizte obrázek 8).

CORD využívá SDN, síťové virtualizace a cloudových technologií k vybudování agilních datových center na hranici sítí operátorů. Využití SDN architektury umožňuje snížení ceny zařízení, programovatelnost řídící vrstvy a otevřenost, která podporuje inovaci. Síťová virtualizace umožňuje škálovatelnost přesunu sítě z fyzického hardwaru na virtuální stroje a cloud (Subramanian a Voruganti, 2016, s. 163–164).

CORD je v současné době rozdělen do tří distribucí podle trhu, pro který je daná distribuce určena: M-CORD, R-CORD a E-CORD. M-CORD distribuce umožňuje poskytování služeb na hranici mobilních 5G sítí a je doplněna disagregovanou rádiovou přístupovou sítí a open

[45] V případě tranzitní komunikace neobsahuje SDN síť zdroj ani cíl komunikace.
[46] Cesta v síti je vytvořena dříve, než je její existence skutečně vyžadována (před příchodem prvního paketu).
[47] Cesta v síti je vytvořena až v momentě, kdy je její existence vyžadována (po příchodu prvního paketu).
[48] Síťový operátor je poskytovatel kabelových a bezdrátových komunikačních služeb, který vlastní nebo řídí síťovou infrastrukturu (Network Operators, c2019).
[49] *Head-end* je zařízení, které přijímá televizní signál ze satelitů, transformuje ho na signál přenosný přes kabelové sítě a poté provádí jeho distribuci (Headend, c2019).

source mobilním jádrem. R-CORD distribuce umožňuje poskytování ultra-širokopásmových služeb prostřednictvím kabelových přístupových technologií, jako je například Data Over Cable Service Interface Specification (DOCSIS), Gigabit Passive Optical Network (GPON) a další. E-CORD distribuce umožňuje poskytování podnikových služeb, jako je například virtuální privátní síť (VPN) a optimalizace aplikací běžících v prostředí Metropolitan Area Network (MAN) a Wide Area Network (WAN) sítí.



Obrázek 9: CORD architektura

Zdroj: (CORD Platform, c2019)

Platforma CORD využívá desítky open source projektů, včetně čtyř hlavních, které představují její základní architektonické prvky (vizte obrázek 9):

- OpenStack umožňuje vytvářet a zajišťovat virtuální stroje a virtuální sítě.
- Docker umožňuje spouštět instance prvků služeb v izolovaných kontejnerech.
- ONOS kontrolér slouží k řízení fyzické infrastruktury a virtuálních sítí, které jsou na ni namapovány. Umožňuje efektivní poskytování služeb koncovým uživatelům.
- Extensible Service Control Plane (XOS) framework umožňuje sestavení a skládání služeb.

4.3 Ostatní funkcionality

V této kapitole budou popsány funkcionality, které byly do ONOS kontroléru zařazeny po květnu 2017 (uvedeny v tabulce 4 v abecedním pořadí), protože starší funkcionality byly již popsány v rámci mé bakalářské práce (Mokráček, 2017). Většina zde popsaných funkcionalit se opět týká poskytovatelů služeb, z větší části se však jedná o funkcionality, které se nachází ve stavu rozpracování. Dále zde bude také popsán modul Reactive Forwarding, neboť se používá v rámci praktické části práce k testování funkcionalit vlastního modulu.

Tabulka 4: Ostatní funkcionality ONOS kontroléru

Funkcionalita	**Název**
Automatic and Real-Time detection and Mitigation System	org.onosproject.artemis
Connectivity Fault Management	org.onosproject.cfm
gRPC	org.onosproject.protocols.grpc, org.onosproject.grpc.nb.service, org.onosproject.grpc.registry
Intent Monitor and Reroute service	org.onosproject.imr
Open and Disaggregated Transport Network	org.onosproject.odtn-api, org.onosproject.drivers.odtn-driver, org.onosproject.odtn-service
P4 a P4Runtime	org.onosproject.protocols.p4runtime, org.onosproject.p4runtime, org.onosproject.drivers.p4runtime
Rabbit MQ	org.onosproject.rabbitmq
Reactive Forwarding	org.onosproject.fwd
Simplified Overlay Network Architecture	org.onosproject.ovsdb-base, org.onosproject.optical-model, org.onosproject.drivers, org.onosproject.drivers.ovsdb, org.onosproject.openflow-base, org.onosproject.openstacknode, org.onosproject.openstacknetworking
Transaction Language 1	org.onosproject.tl1
Transport API	org.onosproject.models.tapi

Zdroj: vlastní zpracování

Automatic and Real-Time detection and Mitigation System (Artemis) je nástroj umožňující správci sítě v reálném čase detekovat a automaticky zmírnit dopad BGP prefix *hijack*[50] incidentu na prefixy sítě. Artemis monitoruje v reálném čase BGP data. Díky tomu dokáže během pár sekund detekovat BGP prefix *hijack* a během pár minut zmírnit jeho dopady (Mavrommatis, 2018).

Connectivity Fault Management (CFM) je ONOS implementace standardu IEEE 802.1ag a International Telegraph Union (ITU) Y.1731. IEEE 802.1ag je standard pro správu selhání

[50] BGP prefix *hijack* znamená, že se útočník vydává za napadenou síť. Zamění prefixy napadané sítě za jiné a ty jsou poté propagovány sousedním sítím pomocí BGP protokolu. Výsledkem může být to, že je provoz předán útočníkovi namísto jeho legitimního cíle, způsobení Denial of Service (DoS) nebo odposlech provozu (Kruse, 2018).

konektivity pro lokální a metropolitní sítě podporující VLAN. Y.1731 k tomuto standardu přidává ještě monitorování výkonu. CFM specifikuje protokoly, postupy a spravované objekty umožňující správu selhání spojení. Pomocí výše zmíněných specifikací je umožněna verifikace a zjišťování cest skrze mosty a Local Area Network (LAN) sítě. Selhání spojení lze díky standardu detekovat a izolovat na konkrétní most či LAN síť (*IEEE 802.1ag-2007*, 2007; Condon, 2017).

gRPC je *northbound* API sloužící ke zprostředkování vysoce efektivní komunikace mezi ONOS a externími aplikacemi. Komunikace je nezávislá na použitém programovacím jazyce. Klientská část gRPC aplikace může přímo volat metody serverové části aplikace (vzdálené volání procedur), jako by se jednalo o její součást. gRPC využívá vyrovnávací paměť protokolu (*protocol buffers*) pro zajištění efektivní serializace pro přenos dat a Hypertext Transfer Protocol (HTTP) protokol pro asynchronní komunikaci. Díky gRPC bude v budoucnu možné přemístit více ONOS aplikací mimo ONOS kontrolér, čímž se sníží spotřeba systémových prostředků. gRPC poskytuje také určitý stupeň izolace. Díky izolaci se snižuje riziko, že chyba v jedné ONOS aplikaci ovlivní celý systém (Boswell, 2017; What is gRPC?, c2019).

Intent Monitor and Reroute service (IMR) umožňuje externím aplikacím monitoring statistik a přesměrování specifických záměrů. Jádro ONOS kontroléru, jak již bylo uvedeno v kapitole 2.4, překládá vytvořené záměry na nízko-úrovňová *flow* pravidla, která jsou aplikována přímo na SDN zařízení v síti. Monitorování statistik tedy znamená získávání statistik ohledně všech nízko-úrovňových *flow* pravidel, na které byl záměr přeložen jádrem ONOS kontroléru. Přesměrování se týká optimalizace jednotlivých záměrů. Externí aplikace může na základě sesbíraných statistik například upravit cestu mezi dvěma koncovými zařízeními tím, že upraví, přidá nebo smaže některá z nízko-úrovňových *flow* pravidel (Sanvito et al., 2018; Moro a Sanvito, 2017).

Open and Disaggregated Transport Network (ODTN) umožňuje vytvoření spojení mezi datovými centry, která se skládají z optických zařízení od různých výrobců. Zařízení různých výrobců budou místo proprietárního softwaru využívat open source software a dodržovat otevřené a společné standardy. Cílem je vytvořit takový open source software, který by umožňoval řídit spojení mezi datovými centry jako celek, nehledě na to, že se skládá ze zařízení od různých výrobců. Výrobci zařízení se tedy mohou soustředit pouze na specifické komponenty jejich zařízení a nikoli na jejich software. To umožňuje rychlejší inovaci a snížení ceny. ODTN je podporován velkými společnosti, jako je například COMCAST, Telefonica, Nokia, Ciena a další. ODTN vychází z otevřených průmyslových standardů, jako je například Transport API a OpenConfig (Open and Disaggregated Transport Network (ODTN), c2019).

Programming Protocol-independent Packet Processors (P4) je Domain-specific language (DSL)[51], který umožňuje specifikovat chování datové vrstvy zařízení. Umožňuje programovat softwarové přepínače, síťové karty založené na programovatelném hradlovém poli neboli Field Programmable Gate Array (FPGA) a přepínače založené na rekonfigurovatelných integrova-

[51] DSL je programovací jazyk vyvinutý ke splnění specifických požadavků. Je zaměřen na omezenou konkrétní problémovou doménu. DSL může být vytvořen tak, aby vyhovoval potřebám konkrétní platformy (Rouse, 2014b).

ných obvodech neboli Application Specific Integrated Circuit (ASIC). P4 umožňuje protokolově nezávislé programování na různých úrovních. Například parsování a modifikaci nových nestandardních hlaviček různých komunikačních protokolů a konfiguraci tabulek zařízení. P4 využívá vysokoúrovňové abstrakce a tím umožňuje programování nezávislé na zařízení. P4 programy by měly být přenositelné. Stejný P4 program kompilovaný pro různá zařízení, by měl produkovat stejné chování. P4 dále umožňuje tzv. rekonfigurovatelnost v poli. To znamená, že zařízení nakonfigurované P4 programem může být překonfigurováno novým P4 programem. P4 sice umožňuje programovat chování datové vrstvy zařízení, nicméně neumožňuje její řízení ani konfiguraci za běhu (Boswell a Cascone, 2019).

K řízení a konfiguraci datové vrstvy zařízení, definované pomocí P4 programu, slouží P4Runtime protokol. Příkladem řízení a konfigurace může být vložení pravidla do tabulky zařízení, jejíž vlastnosti byly definovány P4 programem. P4 runtime protokol je *southbound* protokolem a nachází se ve vrstvě *southbound* protokolů – vizte obrázek 2 (Cascone, 2018).

Rabbit Message Queue (MQ) umožňuje zachytávání a parsování zpráv a notifikací kontroléru ONOS. Zachycenou ONOS událost/zprávu (například připojení nového zařízení) konvertuje Rabbit MQ do dokumentu ve formátu JSON. Dokument je poté publikován na MQ serveru. Z MQ serveru mohou být dokumenty odebírány ONOS a externími aplikacemi (ADARA Networks, 2016).

Reactive Forwarding je modul umožňující reaktivní směrování paketů. Modul neumožňuje Layer 3 (L3) směrování[52], nicméně nelze říci, že by se jednalo o modul implementující pouze logiku Layer (L2) přepínaní. Směrování pomocí Reactive Forwarding modulu totiž nemusí být založeno, na rozdíl od klasického L2 přepínaní, pouze na informacích z linkové vrstvy (MAC adresa a port). Modul umožňuje nakonfigurovat i další kritéria shody, kromě výchozí zdrojové/cílové MAC adresy podporuje: zdrojovou/cílovou IP adresu verze 4, typ Internet Control Message Protocol (ICMP), ICMP kód, zdrojový/cílový port TCP/User Datagram Protocol (UDP) protokolu, zdrojovou/cílovou IP adresu verze 6 a VLAN ID. Dále umožňuje konfiguraci priority a časového limitu pro vymazání nepoužívaného *flow* pravidla. Konfigurace se provádí pomocí CLI ONOS kontroléru. Modul sestavuje cestu ze zdroje k cíli postupně ve spolupráci s globálním pohledem na topologii, který je dostupný skrze službu *TopologyService*. Každé zařízení na cestě předává neznámý paket[53] ONOS kontroléru a Reactive Forwarding určuje, na který port bude paket směrován. Tímto způsobem se paket dostane skrze síť až k cíli. Díky službě *TopologyService*, resp. pohledu na kompletní topologii se nemusí modul starat o smyčky v topologii. Služba *TopologyService* mu vždy poskytne nejlepší další spojení na cestě, pokud takové existuje (Radoslavov, 2015).

Simplified Overlay Network Architecture (SONA) umožňuje pronajímat jednotlivým zákazníkům virtuální sítě optimalizované pro cloudově orientovaná datová centra. Virtuální síť si vytváří zákazník sám (bez zásahu poskytovatele služby) pouze pro sebe (není sdílena s ostatními zákazníky). Zákazník nemá povědomí o mapování vytvořené virtuální sítě na fyzickou.

[52] L3 směrování umožňuje směrování paketů mezi zařízeními, která se nacházejí v různých sítích.
[53] Paket, pro který neexistuje pravidlo ve *flow* tabulce.

SONA je v podstatě soubor ONOS aplikací, které dohromady poskytují ovladač pro OpenStack Neutron Modular Layer 2 (ML2)[54] a L3 směrování. (Moon a Li, 2018; What is OpenStack?, 201?; Swanson, 2016).

Transaction Language 1 (TL1) je *southbound* protokol určený primárně ke správě optických zařízení. TL1 je Man Machine Language (MML). MML značí, že zprávy TL1 jsou dobře čitelné a zapisovatelné jak pro zařízení, tak i pro člověka. Zprávy protokolu TL1 umožňují správci či operačnímu systému (ONOS) správu síťových zařízení. Správa zařízení je prováděna pomocí TL1 příkazů, které jsou posílány z TL1 kontroléru TL1 zařízením ve formě zpráv. ONOS kontrolér implementuje rozhraní, která správu pomocí TL1 protokolu umožňují. Například implementace rozhraní TL1 kontroléru, která udržuje kolekci TL1 zařízení. Kolekce TL1 zařízení umožňuje jejich připojení, posílání a příjem zpráv a registraci posluchačů (De Leenheer a Campanella, 2017; Transactional Language 1, 198?).

Transport API (TAPI) je další typ *northbound* API ONOS kontroléru. Jedná se o standardizované API definované ONF, které umožňuje TAPI klientovi získání informací z domény transportních síťových zařízení a její řízení. TAPI klientem může být například externí aplikace. Získávání informací a řízení domény síťových zařízení je prováděno skrze TAPI server (ONOS kontrolér). TAPI podporuje jak vysoko-úrovňové služby nezávislé na technologii (tvorba pomocí záměrů), tak i služby jejichž specifikace závisí na technologii. TAPI umožňuje programovatelné řízení transportních sítí. Programovatelné řízení umožňuje rychlejší a flexibilnější alokaci síťových zdrojů, které umožňují splnění požadavků aplikací (šířka pásma, latence). Mezi výhody využívání TAPI patří: snížení nákladů v důsledku usnadnění řízení sítě, redukce zpoždění při zavádění nových zařízení a služeb a možnost vytvářet a nabízet nové služby, například virtualizace pro 5G a Internet of Things (IoT) aplikace (Transport API (TAPI) 2.0 Overview, 2017).

[54] Neutron ML2 je framework, který umožňuje cloudovému operačnímu systému OpenStack současně využívat různé technologie L2 vrstvy – Virtual Extensible LAN (VxLAN), *flat*, Generic Routing Encapsulation (GRE), VLAN (Neutron/ML2, 201?).

5 VÝVOJ MODULŮ PRO ONOS KONTROLÉR

Kapitola, pokud není uvedeno jinak, čerpá z části oficiální dokumentace ONOS kontroléru (Koshibe a Lantz, 2017).

K vývoji *server side* logiky vlastního modulu se používá, stejně jako pro vývoj ONOS kontroléru, vysokoúrovňový objektově orientovaný programovací jazyk Java. K implementaci business logiky modulu, resp. síťových funkcionalit poskytuje ONOS kontrolér dobře dokumentované Java APIs (dokumentaci lze nalézt na oficiálních stránkách[55]). Vlastní modul, stejně jako samotný ONOS kontrolér, představuje Maven projekt, který se skládá z kolekce tříd. Ty jsou spojeny pomocí XML souboru Project Object Model (POM)[56]. Maven umožňuje z celého projektu vytvořit OSGi svazek, který je poté načítán do OSGi kontejneru (Apache Karaf). Vlastní modul je také možné integrovat do všech přístupových rozhraní ONOS kontroléru – GUI, REST API a ONOS CLI. K vývoji GUI, resp. *client side* logiky modulu se používá javascriptový framework AngularJS verze 1.3.5. Níže bude uvedeno, jak postupovat při samotném vývoji modulu. Před prováděním níže uvedeného postupu je nutné nejprve zprovoznit verzi ONOS kontroléru určenou pro vývoj (vizte kapitola 3).

5.1 Obecný postup při vývoji modulu

V této podkapitole bude popsáno, jak postupovat při vývoji vlastního modulu pro ONOS kontrolér. Podkapitola bude rozdělena do čtyř částí: generování struktury modulu, úprava a popis vlastností modulu v souboru *pom.xml*, práce s vývojovým prostředím (podporovaná vývojová prostředí, vzdálené ladění kódu) a nasazení modulu.

Generování struktury modulu

Ke generování struktury modulu, včetně přístupových rozhraní, se používá utility skript *onos-create-app*, který využívá Maven archetypů[57]. (Kromě generování se dále Maven používá také k buildu vlastních modulů.) Nejprve je tedy zapotřebí pomocí následujícího příkazu nainstalovat Maven:

```
sudo apt install maven
```

Před samotným generováním je nutné specifikovat verzi ONOS kontroléru, pro kterou bude modul určen. Nicméně pro všechny verze nejsou archetypy dostupné. Kompletní výčet dostupných archetypů, lze nalézt v Maven repositáři[58]. Pokud pro používanou verzi ONOS kontroléru

[55] http://api.onosproject.org/1.15.0/apidocs/

[56] POM představuje objektový model Maven projektu. Jedná se o XML reprezentaci celého Maven projektu, která je uložena v kořenovém adresáři projektu pod názvem *pom.xml*. Tento soubor obsahuje veškeré potřebné informace o projektu (identifikátor, závislosti atd.) a konfiguraci, kterou Maven používá při jeho buildu (POM Reference, c2002-2019).

[57] Archetyp je původní vzor nebo model, ze kterého jsou vytvořeny všechny ostatní věci stejného druhu. Maven archetypy tedy umožňují vytvářet nové Maven projekty, které sdílí obecnou strukturu. Vývojáři se díky tomu mohou věnovat business funkcionalitám a nemusí opakovaně vytvářet a konfigurovat základní strukturu, kterou mají projekty společnou (Introduction to Archetypes, c2002-2019).

[58] https://mvnrepository.com/artifact/org.onosproject/onos-archetypes

není archetyp dostupný, je doporučeno použít jeho nejbližší nižší verzi. Následující příkaz umožňuje zjištění používané verze ONOS kontroléru (zadává se do ONOS CLI):

```
summary
```

Následující příkaz slouží ke specifikování verze ONOS kontroléru (zadává se v terminálu, který se bude používat ke generování struktury):

```
export ONOS_POM_VERSION=VERZE
```

Pokud byla správně nastavena cesta ke kořenovému adresáři ONOS kontroléru (vizte kapitola 3.1) a zároveň zvolena dostupná verze archetypu, měl by být utility skript *onos-create-app* plně funkční a dostupný z libovolného adresáře. Následující příkaz vytvoří v aktuálním adresáři nový adresář, do kterého vygeneruje základní strukturu modulu, kterou je možné zkompilovat a nasadit na ONOS kontrolér (všechny parametry jsou povinné):

```
onos-create-app app GROUPID ARTIFACTID VERZE-SNAPSHOT PACKAGE
```

Parametry:

- `groupId` slouží k unikátní identifikaci projektu mezi všemi Maven projekty v rámci Maven repositáře. Pojmenování by mělo být v souladu s konvencemi pro pojmenování Java balíčků. To znamená, že by *GroupId* mělo být ve formátu reverzní DNS notace (např. *org.foo*).
- `artifactId` definuje jméno spustitelného *.jar* souboru a adresáře, do kterého bude vygenerována struktura modulu. *ArtifactId* by se měl skládat pouze z malých písmen a pomlček (např. *dp-app*).
- `verze` definuje aktuální verzi modulu. *Verze* může obsahovat pouze čísla oddělená tečkami (např. *1.0.0*).
- `package` definuje jméno balíčku, do kterého bude vygenerován kód modulu (*server side* logika). Název balíčku by měl být v souladu s konvencemi pro pojmenování Java balíčků a měl by obsahovat *GroupId* (např. *org.foo.app*).

Vygenerování základní struktury modulu lze také provést v interaktivním režimu, kdy je programátor postupně dotazován na parametry uvedené u předchozího příkazu. Nevýhodou tohoto postupu je fakt, že je zapotřebí ještě dodatečně upravit soubor *pom.xml*. Jinak při buildu modulu nedojde k vytvoření souboru OAR, který umožňuje jeho nasazení na ONOS kontrolér. Následující příkaz slouží k vygenerování základní struktury modulu v interaktivním režimu:

```
onos-create-app
```

Po vygenerování základní struktury modulu, je možné vygenerovat rozhraní, která budou integrována do přístupových rozhraní ONOS kontroléru. Všechny příkazy sloužící ke generování rozhraní musí být prováděny v adresáři s vygenerovanou základní strukturou a zároveň mít stejné hodnoty parametrů jako příkaz sloužící k jejímu generování. Následující příkaz slouží k vygenerování struktury umožňující implementaci vlastního ONOS CLI příkazu:

```
onos-create-app cli GROUPID ARTIFACTID VERZE-SNAPSHOT PACKAGE
```

Následující příkaz slouží k vygenerování struktury umožňující implementaci vlastního REST API:

```
onos-create-app rest GROUPID ARTIFACTID VERZE-SNAPSHOT PACKAGE
```

Následující příkazy slouží k vygenerování struktury umožňující implementaci dvou typů pohledů, které jsou plně integrovány do GUI ONOS kontroléru (*ui* – obecný pohled obsahující několik tlačítek s ukázkovou funkcionalitou, hodící se například pro nastavení modulu a *uitab* – pohled obsahující skelet tabulky vhodný například k analýze datového provozu):

```
onos-create-app ui GROUPID ARTIFACTID VERZE-SNAPSHOT PACKAGE
onos-create-app uitab GROUPID ARTIFACTID VERZE-SNAPSHOT PACKAGE
```

Úprava pom.xml

Po vygenerování struktury včetně přístupových rozhraní je nezbytné provést úpravu souboru *pom.xml*. Zejména je zapotřebí provést úpravu vlastností (*properties*) modulu, které se nachází mezi XML tagy *properties* (vizte kompletní ukázka vlastností modulu v příloze A). Tyto vlastnosti mohou být zakomentované (pokud byl použit při generování interaktivní režim), a tím pádem při buildu nebude vytvořen potřebný soubor OAR, který umožňuje nasazení modulu na ONOS kontrolér. V každém případě však budou obsahovat výchozí hodnoty, které je nutné změnit. Při úpravě by měla být věnována pozornost zejména následujícím vlastnostem:

- `onos.app.title` představuje název modulu, který bude zobrazen v GUI ONOS kontroléru (záložka „Applications“) a ONOS CLI.
- `api.description` představuje popis vlastního REST API modulu, který lze nalézt v automaticky generované dokumentaci.
- `api.title` představuje název REST API vlastního modulu. Díky názvu je možné REST API vlastního modulu odlišit od ostatních, která ONOS kontrolér poskytuje.
- `web.context` představuje část identifikátoru zdroje REST API vlastního modulu. Musí být zadáván ve tvaru */onos/ARTIFACTID*.
- `onos.app.requires` umožňuje definovat ONOS aplikace, které daný modul vyžaduje k provádění svých vlastních síťových funkcionalit. Jednotlivé ONOS aplikace se uvádějí pomocí názvu v reverzní DNS notaci a jsou odděleny čárkou. Pokud jsou uvedené ONOS aplikace pouze nainstalovány, tak je ONOS kontrolér automaticky aktivuje společně s daným modulem. V případě, že nejsou všechny požadované ONOS aplikace nainstalovány, není provedena aktivace modulu. ONOS kontrolér pouze vypíše ve svém logu, které z požadovaných ONOS aplikací nejsou dostupné.

Práce s vývojovým prostředím

Pro pohodlnější vývoj modulu je zapotřebí vygenerovanou strukturu naimportovat do vhodně zvoleného vývojového prostředí. Oficiální stránky ONOS kontroléru sice doporučují vývojové prostředí IntelliJ IDEA, nicméně je možné použít jakékoli vývojové prostředí, které podporuje Maven a vzdálené ladění kódu (např. Eclipse, NetBeans a další). Konfigurace vzdáleného ladění kódu a způsob importování projektu zde popisován nebude, neboť se u různých vývojových prostředí může lišit. Pro každé vývojové prostředí však platí, že pro připojení *debuggeru*

k ONOS kontroléru se používá IP adresa (v případě lokálního vývoje *localhost*) konkrétní instance a port 5005. Vzdálené ladění kódu lze provádět pouze v případě, že je modul aktivován na instanci ONOS kontroléru, která byla spuštěna s parametrem *debug*.

Nasazení modulu

Nejprve je zapotřebí provést build modulu, který bohužel není možné provádět přímo ve zvoleném vývojovém prostředí. Je k tomu zapotřebí přímo použít buildovací nástroj Maven. Následující příkaz slouží k buildu modulu a jeho instalaci do lokálního Maven repositáře (příkaz je nutné provádět v kořenovém adresáři modulu):

```
mvn clean install
```

K nasazení (instalace a aktivace) modulu na ONOS kontrolér se na rozdíl od verze pro produkční prostředí primárně používá utility skript *onos-app*, který je stejně jako *onos-create-app* dostupný z libovolného adresáře. K instalaci a aktivaci modulu je samozřejmě také možné použít postup popsaný v kapitole 3.4. Nicméně tento postup je z hlediska vývoje nepraktický, neboť se skládá z více kroků a jeho použití je zdlouhavé. Následující příkaz umožňuje provést instalaci a aktivaci modulu v jednom kroku bez ohledu na to, zdali je modul již na ONOS kontroléru nasazen (příkaz je nutné provádět v kořenovém adresáři modulu, v případě lokálního vývoje lze místo IP adresy ONOS instance použít *localhost*):

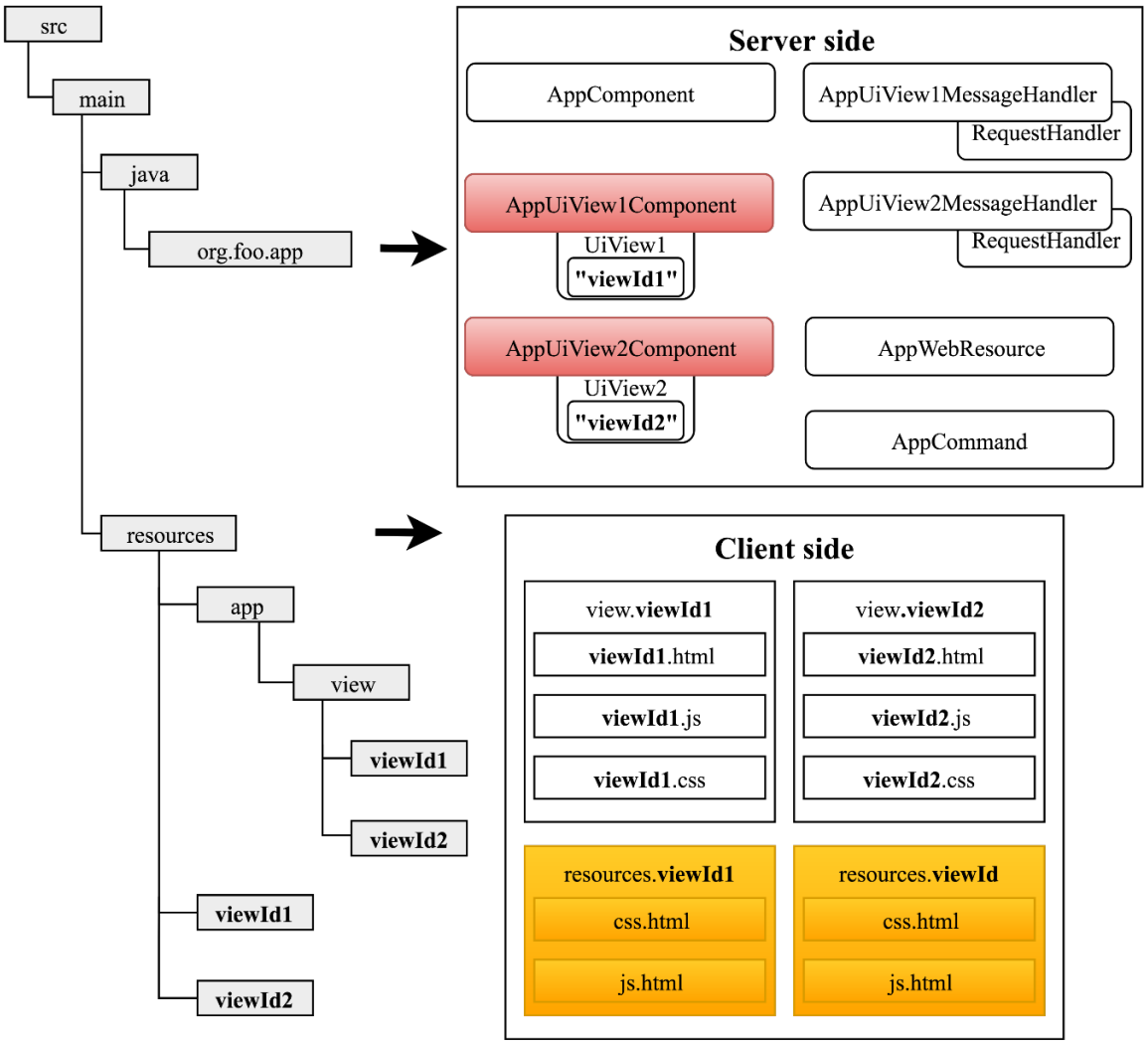
```
onos-app IP_ADRESA_ONOS_INSTANCE reinstall! target/NAZEV_ONOS_OAR_SOUBORU.oar
```

Postup nasazení modulu lze ještě zjednodušit spojením dvou výše uvedených příkazů do jednoho následujícím způsobem (příkaz je nutné provádět v kořenovém adresáři modulu):

```
mvn clean install && onos-app IP_ADRESA_ONOS_INSTANCE reinstall!
target/NAZEV_ONOS_OAR_SOUBORU.oar
```

5.2 Popis a úprava vygenerované struktury

Tato kapitola bude věnována popisu a úpravě struktury modulu, která je znázorněna na obrázku 10. Součástí popisu některých částí budou také ukázky jejich zdrojových kódů a popis používaných anotací v kontextu OSGi frameworku. Úprava struktury se týká zejména částí, které slouží k implementaci *server* a *client side* logiky pohledů integrovaných do GUI ONOS kontroléru (na obrázku 10 znázorněny červenou a žlutou barvou).



Obrázek 10: Vygenerovaná struktura modulu

Zdroj: vlastní zpracování

Třída ***AppComponent*** představuje hlavní třídu modulu, která slouží zejména k implementaci jeho business funkcionalit (např. zpracování paketů). Z hlediska OSGi frameworku se jedná o tzv. komponentu[59], a proto musí být vždy anotována anotací *@Component*. Anotace *@Component* umožňuje třídě *AppComponent* definovat pomocí anotace *@Property* konfigu-

[59] Komponenta představuje objekt, jehož životní cyklus je spravován OSGi kontejnerem (Thangam, 2015).

rovatelné parametry modulu (vizte kapitola 2.5.2 a 3.4) a pomocí anotace *@Reference* injektovat služby (poskytovány jádrem ONOS kontroléru a ostatními moduly) umožňující implementaci požadovaných business funkcionalit. S anotací *@Component* dále souvisí metody anotované anotací *@Activate*, resp. *@Deactivate*, které jsou volány automaticky na začátku, resp. na konci životního cyklu modulu. V rámci metody volané na začátku životního cyklu lze například registrovat modul (získání unikátního *appId* – vizte kapitola 1.1.1) či jeho konfigurovatelné parametry u OSGi kontejneru (Apache Karaf). V rámci metody volané na konci životní cyklu by se měl provádět „úklid" (např. zrušení registrace konfigurovatelných parametrů či odstranění *flow pravidel* vytvořených modulem).

Anotace *@Service* je volitelná a slouží k označení, že anotovaná komponenta (klasická třída nemůže být službou) je v kontextu OSGi frameworku zároveň i službou[60]. Díky tomu je možné používat veřejné metody třídy *AppComponent* v ostatních třídách modulu nebo dokonce i v jiných modulech.

```
// (immediate = true) instance komponenty je vytvořena okamžitě po její aktivaci
@Component(immediate = true)
// value definuje rozhraní služby, v tomto případě je to sama třída
@Service(value = AppComponent.class)
public class AppComponent {
     // Kardinalita určuje, zdali se očekává 0..N služeb daného typu
     // V níže uvedených případech se očekává přesně jedna služba (1..1)
     @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
     protected CoreService coreService;
     @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
     protected ComponentConfigService configurationService;
     private final int DEFAULT_VALUE = 5;
     // definice konfigurovatelného parametru
     @Property(name = "propExample", intValue = DEFAULT_VALUE, label = "desc")
     private int propExample = DEFAULT_VALUE;

     private final Logger log = LoggerFactory.getLogger(getClass());
     private ApplicationId appId;
     @Activate
     protected void activate() {
          // bez appId nelze používat některé služby
          appId = coreService.registerApplication("org.foo.app");
          configurationService.registerProperties(getClass());
          log.info("Started");
     }
     @Deactivate
     protected void deactivate() {
          configurationService.unregisterProperties(getClass(), false);
          log.info("Stopped");
     }
}
```

[60] Služba představuje v kontextu OSGi framworku takovou komponentu, která může být injektována do jiné komponenty. Komponenta, do které je služba injektována může využívat veřejné metody poskytované službou (Thangam, 2015).

Třída ***AppCommand*** slouží k implementaci vlastních ONOS CLI příkazů (na každý vlastní příkaz je zapotřebí vytvořit další třídu). Třída musí představovat rozšíření třídy *AbstractShellCommand* a zároveň být anotována dvěma anotacemi – *@Service* umožňující injekci příkazu do kontejneru Apache Karaf a *@Command* sloužící k definici jmenného prostoru[61], jména a popisu příkazu. Poslední povinnou částí je přetížená metoda *doExecute*, jejíž kód se provede po každém zavolání příkazu. Uvnitř třídy lze také pomocí anotace *@Option*, resp. *@Argument* definovat parametry, resp. argumenty příkazu (argument představuje povinnou část příkazu a parametr nepovinnou). Nicméně je nutné si vybrat, zdali bude mít příkaz parametry nebo argumenty, obojí najednou mít nemůže. Z tohoto důvodu je v ukázce zdrojového kódu argument zakomentován.

```
@Service
// Jmenný prostor onos umožňuje volaní příkazu přímo z ONOS CLI bez specifikace
// jmenného prostoru přímo jménem (sample)

// Pokud je zvolen jiný jmenný prostor je zapotřebí volat příkaz následujícím
// způsobem - jmenny_prostor:jmeno (onos:sample)
@Command(scope = "onos", name = "sample",
         description = "Sample Apache Karaf CLI command")
public class AppCommand extends AbstractShellCommand {
     // volání s parametrem: sample -opt HODNOTA_PARAMETRU
     @Option(name = "-opt", aliases = "--opt_long_name",
             description = "desc_example",
             required = false, multiValued = false)
     private String opt = null;
     // index představuju pozici daného argumentu vzhledem k ostatním
     // volání s argumentem: sample HODNOTA_ARGUMENTU_INDEX0
     // @Argument(index=0,name="arg", required=true,
     //           description="desc_example")
     // private String arg = null;

     @Override
     protected void doExecute() {
          if (opt != null) {
               print("Opt value %s", opt);
          } else {
               print("Hello %s", "World");
          }
     }
}
```

Třída ***AppWebResource*** slouží k implementaci REST API modulu, které je plně integrováno do REST API ONOS kontroléru. Vlastní REST API lze použít kromě komunikace s externími aplikacemi také v rámci *client side* logiky GUI modulu. Architektura GUI ONOS kontroléru (*single page application*) totiž neumožňuje zasílání požadavku od klienta, jehož velikost přesahuje cca 500 KiB (problém například v případě nahrávání souborů).

[61] Apache Karaf seskupuje příkazy podle jmenných prostorů. Pro každý jmenný prostor je vytvořena separátní instance příkazového procesoru (Using the console, c2018).

```
// specifikace URL zdrojové třídy – třída musí mít alespoň jednu operaci
// anotovanou anotací označující použitou HTTP metodou
@Path("sample")
public class AppWebResource extends AbstractWebResource {

     // specifikace URL operace
     @Path("greeting")
     // použitá HTTP metoda {GET, POST, PUT, DELETE}
     @GET
     // @Consumes(MediaType.TYP) v případě, že vstupem dané operace je soubor
     public Response getGreeting() {
          ObjectNode node = mapper().createObjectNode().put("hello", "world");
          return ok(node).build();
     }
}
```

Třídy ***AppUiView2MessageHandler*** a ***AppUiView2MessageHandler*** představují rozšíření třídy *UiMessageHandler*. Slouží k implementaci obslužných rutin (*RequestHandler*) veškerých požadavků od jednoho konkrétního pohledu (*client side*), který je injektován do GUI ONOS kontroléru. Každý typ požadavku od klienta je svázán s konkrétní instancí třídy *RequestHandler* pomocí svého typu. Typ požadavku je společně s jeho tělem (*payload*) přenášen na server ve formátu JSON souboru, kde je podle něj vybraná konkrétní instance třídy *RequestHandler*. Následně je požadavek zpracován metodou *process* a klientovi je vrácena odpověď ve formátu JSON souboru.

```
public class AppUiView1MessageHandler extends UiMessageHandler {
     // typ požadavku – musí být použit i v client side logice pohledu
     private static final String EXAMPLE_REQUEST = "ExampleRequest";
     private static final String EXAMPLE_RESPONSE = "ExampleResponse";
     private static final String MESSAGE = "message";
     // kolekce veškerých obslužných rutin určitého pohledu
     @Override
     protected Collection<RequestHandler> createRequestHandlers() {
          return ImmutableSet.of(
               new SampleCustomDataRequestHandler()
          );
     }
     // typ požadavku představuje atribut každé instance RequestHandler
     private final class ExampleRequestHandler extends RequestHandler {
          private SampleCustomDataRequestHandler() {
               super(EXAMPLE_REQUEST);
          }
          @Override
          public void process(ObjectNode payload) {
               ObjectNode result = objectNode();
               result.put(MESSAGE, "Example message");
               sendMessage(EXAMPLE_RESPONSE, result);
          }
     }
}
```

Třídy ***AppUiView1Component*** a ***AppUiView2Component*** je z hlediska udržitelnost kódu výhodné sloučit do jediné třídy ***AppUiComponent*** (vizte obrázek 11), jejíž kompletní zdrojový kód je dostupný v příloze B. Třída *AppUiComponent* představuje, stejně jako třída *AppComponent*, z hlediska OSGi frameworku komponentu a slouží k injekci vlastních pohledů do GUI ONOS kontroléru. Injekce se provádí registrací vlastního rozšíření *UiExtension* u služby *UiExtensionService*, která je poskytována jádrem ONOS kontroléru. Registrace vlastního rozšíření se provádí v rámci metody anotované anotací *@Activate*. (Zrušení registrace se provádí v rámci metody anotované anotací *@Deactivate*.) Instance vlastního rozšíření *UiExtension* obsahuje:

- unikátní identifikátor,
- továrnu *UiMessageHandlerFactory* generující na vyžádání pro konkrétní pohled instanci třídy *UiMessageHandler* (jeden pohled rovná se jedna instance),
- jednu nebo více instancí třídy *UiView* udržující unikátní identifikátor[62], kategorii a název pohledu
- a cestu k souborům *css.html* a *js.html*[63], jejichž obsah je injektován do souboru *index.html* GUI ONOS kontroléru.

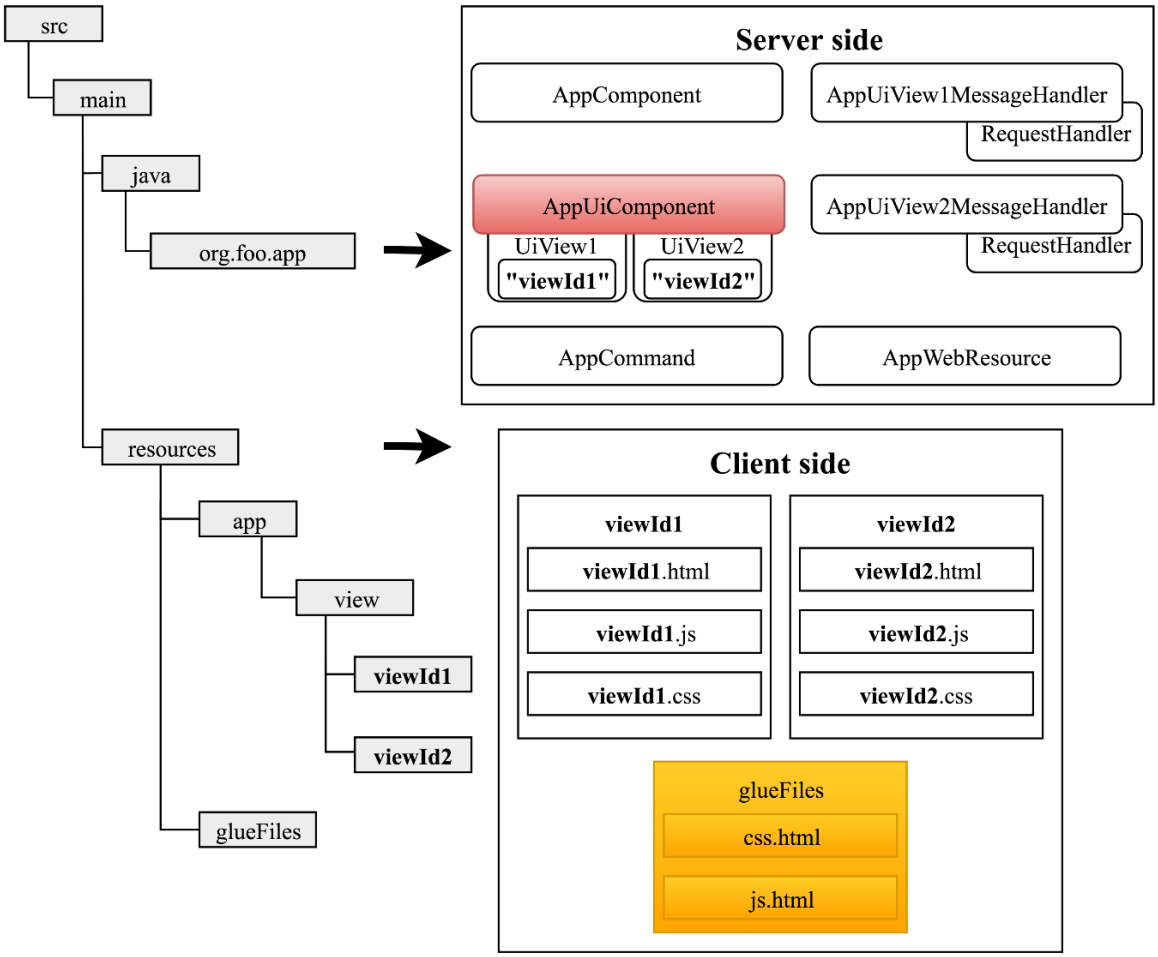
Pokud je provedeno výše zmíněné sloučení tříd *AppUiView1Component* a *AppUiView2Component* do jediné třídy *AppUiComponent*, tak je ještě zapotřebí provést sloučení obsahu *css.html* a *js.html* souborů obou pohledů (celkem čtyři soubory) do dvou souborů stejného typu (vizte příklad níže). Následně je zapotřebí vytvořit v adresáři *resources* adresář *glueFiles* (vizte obrázek 11), vložit do něj oba soubory a specifikovat cestu k tomuto adresáři ve třídě *AppUiComponent* způsobem uvedeným v příloze B na 27. řádku. Sloučení výše uvedených souborů a tříd značně sníží duplicity ve zdrojovém kódu a usnadní případné přidávání dalších pohledů (kompletní struktura po výše zmíněných úpravách je znázorněna na obrázku 11).

```
Příklad css.html:
<link rel="stylesheet" href="app/view/viewId1/viewId1.css">
<link rel="stylesheet" href="app/view/viewId2/viewId2.css">
<link rel="stylesheet" href="app/view/viewId2/bootstrap.min.css">

Příklad js.html:
<script src="https://ajax.googleapis.com/ajax/../jquery.min.js"></script>
<script src="app/view/viewId1/viewId1.js"></script>
<script src="app/view/viewId2/viewId2.js"></script>
```

[62] Unikátní identifikátor pohledu se prolíná celou strukturou modulu (vizte obrázek 11). Veškeré jmenné konvence týkající se *viewId* lze nalézt na oficiálních stránkách ONOS kontroléru – https://wiki.onosproject.org/display/ONOS/Web+UI+Tutorial+-+Creating+a+Custom+View.

[63] Jedná se o tzv. *glue files*, které umožňují vložení Cascading Style Sheets (CSS) a JavaScriptu jednotlivých pohledů.



Obrázek 11: Struktura modulu po úpravě

Zdroj: vlastní zpracování

6 VLASTNÍ MODUL: TRAFFIC ANALYZER

Cílem praktické části práce bylo vyvinout vlastní modul pro ONOS kontrolér umožňující uživateli monitorování datových toků[64] v síťové topologii s možností ručního povolení či zakázání komunikace. Při vlastním vývoji byla ještě navíc přidána funkcionalita umožňující uživateli zasílat každý paket daného datového toku na ONOS kontrolér kvůli uložení jeho dat (*payload*). Uložená data posledního přijatého paketu si uživatel může stáhnout k hlubší analýze. Jedná se o funkcionalitu, na jejímž základě by mohla být založena Layer 7 (L7) inspekce[65]. Každý unikátní datový tok je uložen do kolekce, která umožňuje uživateli monitoring a provádění požadovaných operací. Kromě výše uvedených hlavních funkcionalit umožňuje modul uživateli například uložení kolekce datových toků do souboru a její opětovné načtení (včetně mapování kolekce na síťovou topologii), periodické vytváření záloh a nastavení výchozí politiky (zakázání, povolení, L7 inspekce) při uložení datového toku do kolekce. Veškeré funkcionality, které modul poskytuje jsou dostupné skrze GUI modulu a monitorování datových toků je navíc podporováno i v CLI ONOS kontroléru. Obě uživatelská rozhraní jsou plně integrována do ONOS kontroléru.

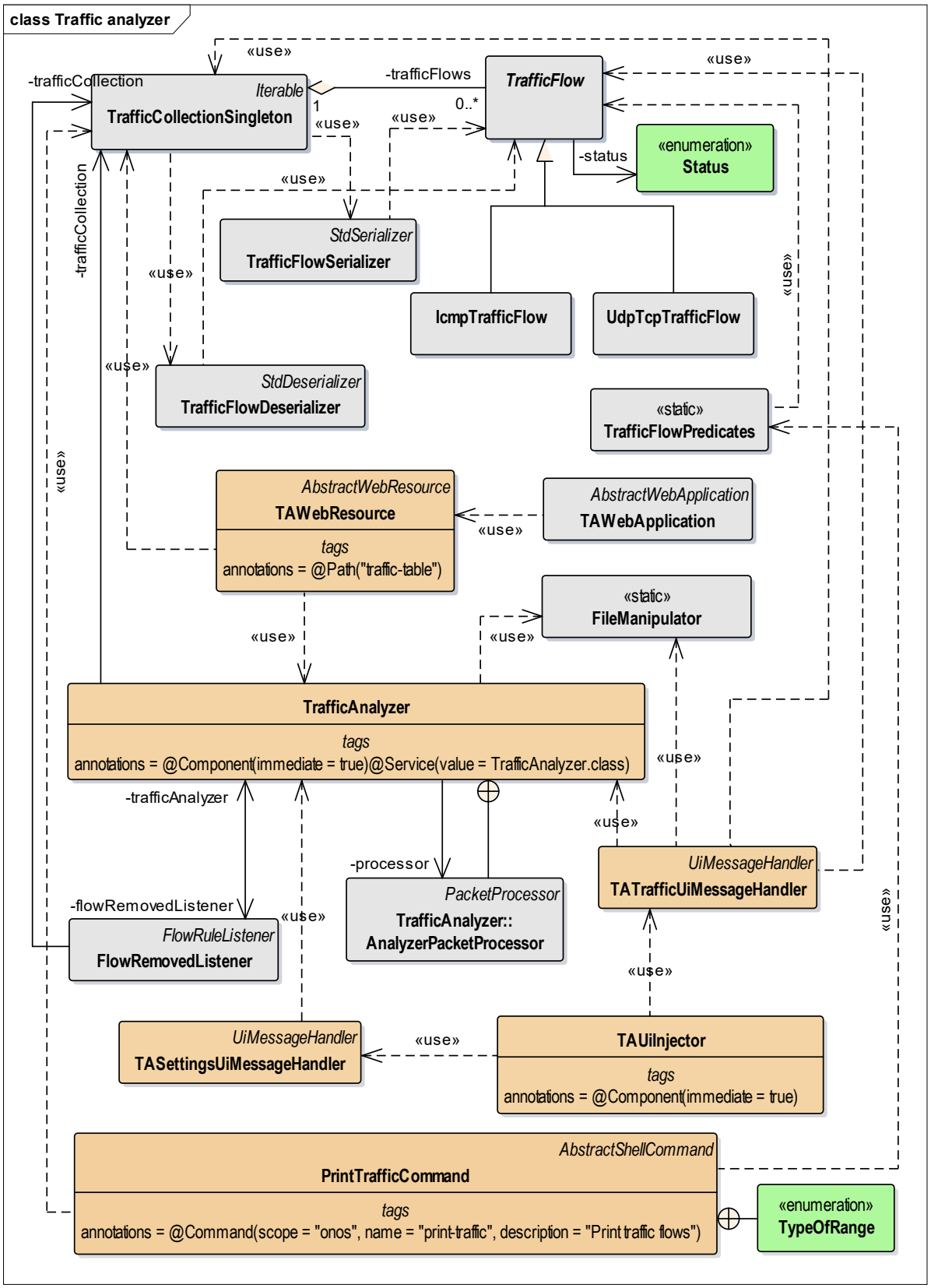
Modul Traffic Analyzer byl vytvořen pro verzi ONOS kontroléru 1.13.2 (Nightingale). Poslední verze zpětně kompatibilní s moduly pro verzi 1.13.2 je verze 1.14.0 (Owl). Na novějších verzích ONOS kontroléru již není možné modul Traffic Analyzer aktivovat. Oproti technologiím používaným k vývoji vlastních modulů pro ONOS kontrolér, které byly uvedeny v úvodu do kapitoly 5, byly navíc použity některé části[66] CSS frameworku Bootstrap, který umožňuje vývoj plně responsivního a *mobile-first*[67] GUI.

6.1 Popis jednotlivých tříd

Na obrázku 12 je zobrazen Unified Modeling Language (UML) diagram modulu Traffic Analyzer. Zlatou barvou jsou na diagramu znázorněny vygenerované třídy, které byly již výše obecně popsány, zelenou barvou výčtové typy a světle šedou barvou zbývající třídy. Z důvodu přehlednosti byly z diagramu vypuštěny jmenné prostory (označující, do kterého balíčku daná třída spadá), atributy a operace jednotlivých tříd a vnitřní třídy implementující *server side* logiku jednotlivých uživatelských požadavků a konstrukci objektů reprezentující datové toky. UML diagramy jednotlivých balíčků obsahující veškeré třídy, ze kterých se modul Traffic Analyzer skládá, lze nalézt v přílohách C a D. V následujících odstavcích bude uveden popis vybraných tříd. Třídy, které byly již výše obecně popsány budou označeny stejně jako v kapitole 5.1. Toto označení bude uvedeno v závorce za názvem dané třídy.

---

[64] Datový tok v rámci modulu představuje sekvenci paketů putujících od zdroje k cíli za použití konkrétního protokolu, popřípadě zdrojového a cílového portu (pokud je použit některý z protokolů transportní vrstvy).
[65] L7 inspekce neboli *deep packet inspection* je metoda filtrování paketů na základě analýzy dat, která paket přenáší (Rouse, 2017).
[66] Vložení kompletního Bootstrapu do projektu způsobovalo nechtěné změny v GUI ONOS kontroléru.
[67] *Mobile-first* je způsob návrhu uživatelského rozhraní, kdy je nejprve navrženo uživatelské rozhraní pro mobilní zařízení, které je následně přizpůsobeno větším obrazovkám (What is a mobile-first approach? Definition, c2018).



Obrázek 12: UML diagram tříd modulu Traffic Analyzer

Modul Traffic Analyzer umožňuje pouze zpracování IPv4 paketů a podporuje celkem tři druhy datových toků – ICMP, UDP a TCP. Uvedené druhy datových toků mají společnou většinu atributů (např. IP adresu zdroje a cíle, použitý protokol přenosu atd.) a operací (např. změna stavu – blokovaný, povolený a L7 inspekce), které s nimi je možné provádět. Abstraktní třída *TrafficFlow,* ze které jsou odvozeny třídy *IcmpTrafficFlow* a *UdpTcpTrafficFlow,* udržuje společné atributy, poskytuje společné operace a deklaruje abstraktní metody. Instance třídy *IcmpTrafficFlow* reprezentuje ICMP datový tok, u kterého je navíc zapotřebí udržovat ICMP kód a typ. Instance třídy *UdpTcpTrafficFlow* umožňuje reprezentovat jak UDP, tak TCP datový tok, neboť u obou je zapotřebí udržovat navíc pouze číslo zdrojového a cílového portu (na typu portu nezáleží).

Ke konstrukci instancí třídy *IcmpTrafficFlow* a *UdpTcpTrafficFlow* je využíván návrhový vzor Builder[68], který je implementován v rámci vnitřních tříd umístěných v jednotlivých třídách reprezentující datové toky – *TrafficFlow::Builder*, *IcmpTrafficFlow::Builder* a *UdpTcpTrafficFlow::Builder* (vizte příloha D – UML diagram balíčku *datastructures*). Vnitřní třída *TrafficFlow::Builder* představuje, stejně jako třída *TrafficFlow*, abstraktní třídu, od které jsou zbylé vnitřní třídy implementující návrhový vzor Builder odvozeny. Návrhový vzor Builder byl použit z důvodu velkého množství parametrů v konstruktorech, přičemž ne všechny byly vždy povinné. Z tohoto důvodu by jejich vyplňování bylo problematické a nespolehlivé. Objekt typu *Builder* umožňující konstrukci instancí je dostupný přes statickou metodu *build*, která se nachází jak ve třídě *IcmpTrafficFlow*, tak i v *UdpTcpTrafficFlow*. Tato metoda přijímá menší množství parametrů, které jsou u konstrukce dané instance vždy povinné. Pomocí zřetězení metod je možné inicializovat další atributy dané třídy. Celý proces konstrukce je zakončen voláním metody *build*, která vrací instanci konkrétní třídy.

Instance třídy *IcmpTrafficFlow* a *UdpTcpTrafficFlow* jsou udržovány v kolekci, která je zapouzdřena ve třídě *TrafficCollectionSingleton*. Třída *TrafficCollectionSingleton* poskytuje například standardní metody kolekce (např. vložení, odebrání, získání, procházení atd.), převod z/do formátu JSON (nahrání a stažení kolekce) atd. Vzhledem k tomu, že v rámci modulu musí existovat pouze jediná instance této třídy (pouze jedna kolekce), byl použit návrhový vzor Singleton, který umožňuje zabezpečit existenci pouze jediné instance dané třídy a poskytuje k ní globální přístupový bod.

Jako nejvhodnější typ kolekce byla vybrána hashmapa, konkrétně třída *HashMap<K, V>*[69], která je dostupná v JDK. Hashmapa byla vybrána z důvodu efektivity, neboť datové toky je zapotřebí velmi často vyhledávat[70]. Vkládá se pouze při načítání hashmapy ze souboru nebo v případě, že na ONOS kontrolér dorazil paket spadající pod datový tok, který v hashmapě ještě není uložen. Ve všech ostatních případech (např. odebrání *flow* pravidla, příchod paketu již

[68] Slouží k oddělení konstrukce objektu od jeho reprezentace.
[69] *K* představuje datový typ klíče a *V* představuje datový typ ukládaných hodnot.
[70] Konstantní asymptotická složitost vyhledávání (metoda *get*) – O (1).

uloženého datového toku atd.) je zapotřebí datový tok vyhledat a popřípadě s ním provést nějaké operace. Všechny standardní metody kolekce, které třída *TrafficCollectionSingleton* poskytuje, ve skutečnosti volají metody instance třídy *HashMap<String, TrafficFlow>*[71].

Třídy *TrafficFlowSerializer* a *TrafficFlowDeserializer* umožňují vlastní JSON serializaci, resp. deserializaci instancí tříd reprezentující datové toky. Serializace slouží k jejich konverzi do JSON objektů a deserializace představuje proces přesně opačný. Vlastní serializaci, resp. deserializaci bylo nutné implementovat z následujících důvodů:

- Absence implementace JSON serializace a deserializace u některých tříd poskytovaných ONOS kontrolérem (např. *DeviceId*), jejichž instance představují hodnoty atributů tříd reprezentující datové toky.
- Nemožnost provést jednoduchou[72] JSON serializaci, resp. deserializaci instance třídy, která je součástí dědické hierarchie, jejímž vrcholem je abstraktní třída.

Třída *TrafficAnalyzer* (*AppComponent*), jak již bylo uvedeno v kapitole 5.2, představuje hlavní třídu modulu sloužící zejména k implementaci jeho business funkcionalit. K jejich implementaci byly použity následující služby poskytované jádrem ONOS kontroléru:

- *PacketService* umožňující zachycení paketů zaslaných ONOS kontroléru a vysílání paketů vytvořených modulem.
- *FlowRuleService* umožňující práci s autoritativní tabulkou (vizte kapitola 2.6) – vkládání a mazání *flow* pravidel, získávání statistik, sledování změn (registrace posluchačů události).
- *FlowObjectiveService* umožňující programování datové vrstvy nezávisle na konfiguraci *pipeline* konkrétního SDN zařízení (vizte kapitola 2.6).
- *DeviceService* umožňující interakci s SDN zařízeními ve spravované topologii.
- *HostService* umožňující interakci s koncovými zařízeními ve spravované topologii.
- *TopologyService* umožňující získávání informací o spravované topologii. Například získání nejkratší cesty neobsahující smyčky mezi dvěma SDN zařízeními.

Třída *FlowRemovedListener* představuje implementaci rozhraní *FlowRuleListener*, které umožňuje zachytávání veškerých událostí týkajících se *flow* pravidel uložených v autoritativní tabulce. Nicméně v rámci metody *event* sloužící k obsloužení události se reaguje pouze na událost týkající se odebrání *flow* pravidla z SDN zařízení. Reakcí na tento typ události je změna stavu[73] datového toku, pro který bylo smazané *flow* pravidlo určeno. Posluchače události typu *FlowRuleListener* je zapotřebí zaregistrovat u služby *FlowRuleService* pomocí metody

[71] Hashmapa představuje jeden z atributů třídy *TrafficCollectionSingleton* – zapouzdření.
[72] Pomocí Jackson APIs.
[73] Změnou stavu je myšlena změna hodnoty atributu, který indikuje přítomnost *flow* pravidla ve *flow* tabulce SDN zařízení.

addListener. Registrace se provádí v rámci metody anotované anotací *@Activate* ve třídě *TrafficAnalyzer*.

Třída *AnalyzerPacketProcessor* představuje implementaci rozhraní *PacketProcessor*, které umožňuje zpracování paketů zaslaných ONOS kontroléru. V rámci metody *process* sloužící ke zpracování paketů se provádí např. vytváření nových datových toků, blokace paketů či ukládání jejich dat. Třídu implementující rozhraní *PacketProcesor* je zapotřebí přidat do *pipeline* ONOS kontroléru pomocí metody *addProcessor*, kterou poskytuje služba *PacketService*. Přidání se provádí v rámci metody anotované anotací *@Activate* ve třídě *TrafficAnalyzer*.

Statická třída *FileManipulator* obsahuje metodu *saveTableInJsonFormatToFile* umožňující uložit kolekci datových toků ve formátu JSON do souboru a metodu *getTableFromFileInJsonString* umožňující její načtení ze souboru. Jedná se pouze o pomocnou třídu, jejíž metody jsou využívány třídou *TrafficAnalyzer* při vytváření, resp. načítání záloh kolekce datových toků.

Třídy *TASettingsUiMessageHandler* a *TATrafficUiMessageHandler* (*AppUiViewMessageHandler*) slouží k implementaci obslužných rutin veškerých požadavků zaslaných od pohledů (*client side*), které modul poskytuje – Settings a Traffic monitor. Obslužné rutiny jednotlivých požadavků jsou implementovány v rámci vnitřních tříd, které je možné nalézt v příloze C.

Třída *TAWebResource* (*AppWebResource*) slouží k implementaci REST API umožňujícího uživateli nahrání JSON souboru, který obsahuje kolekci datových toků. REST API je použito v rámci *client side* logiky pohledu Traffic monitor. Uživatel tedy nemusí používat k nahrávání souborů žádné externí nástroje (např. Postman). REST API bylo nutné implementovat, protože ONOS kontrolér neumožňuje pohledu zaslání požadavku, jehož velikost přesahuje cca 500 KiB. Pokud je velikost požadavku překročena, tak dochází k selhání celého GUI ONOS kontroléru.

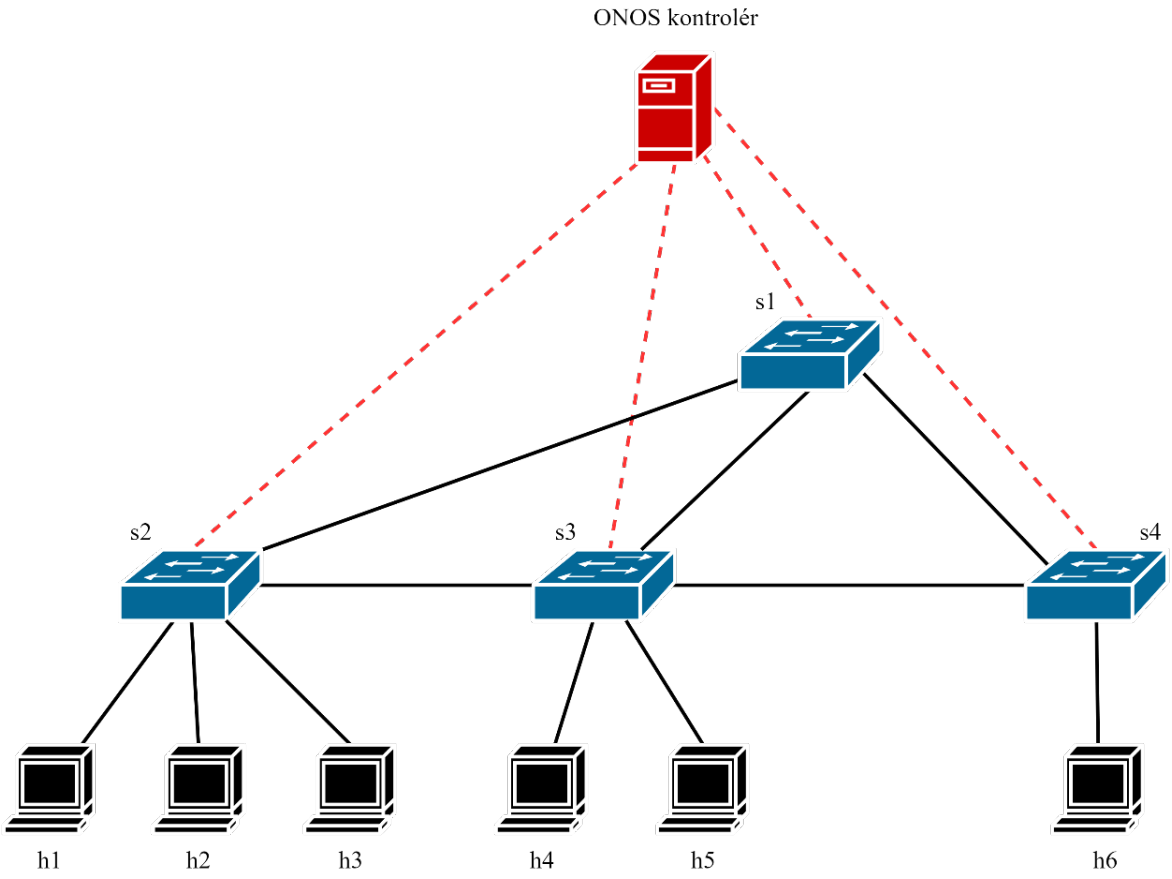
Třída *PrintTrafficCommand* (*AppCommand*) představuje implementaci vlastního příkazu, který umožňuje uživateli monitorování datových toků. V rámci třídy jsou definovány parametry, které uživateli umožňují jejich řazení a filtrování. Parametry umožňující filtrování je možné libovolně kombinovat. Uživatel si tak může například vyfiltrovat pouze datové toky používající specifický zdrojový a cílový port. Pokud je hodnota parametru zadána ve špatném formátu, je do CLI ONOS kontroléru vypsáno chybové hlášení umožňující chybně zadanou hodnotu identifikovat.

Výše zmíněné filtrování datových toků je založeno na predikátech[74] (*Predicate<T>*), které jsou dostupné v JDK. Metody, které umožňují filtrování datových toků založené na predikátech (všechny vrací *Predicate<TrafficFlow>*), jsou implementovány ve statické třídě *TrafficFlowPredicates*. Predikáty, které jednotlivé metody vrací, jsou použity jako vstupní parametr metody *filter*, kterou poskytují tzv. proudy (*streams*).

[74] *Predicate<T>* představuje funkční rozhraní (pouze jedna abstraktní metoda), do kterého je možné přiřadit *lambda* výraz. Rozhraní poskytuje pouze metodu *test*, která má pouze jeden vstupní parametr (typu *T*) a vrací *boolean* (Java 8 Predicate with Examples, 2018).

6.2 Testování funkcionalit

K testování funkcionalit modulu byl použit síťový emulátor Mininet, který umožňuje simulaci libovolné topologie. Jednotlivá SDN zařízení, ze kterých se topologie skládá, jsou připojena k ONOS kontroléru, který provádí programování jejich datové vrstvy (resp. kompletní správu topologie). Pro účely testování byla pomocí grafického nástroje MiniEdit navržena vlastní topologie, která je znázorněna obrázku 13. Navržená topologie záměrně obsahuje smyčky, aby co nejlépe imitovala prostředí reálných sítí, ve kterém se smyčky zpravidla vyskytují z důvodu redundance spojení v případě výpadku.



Obrázek 13: Testovací topologie

Modul Traffic Analyzer provádí směrování komunikace pouze v případě, že uživatel požaduje, aby každý paket daného datového toku byl poslán ONOS kontroléru z důvodu uložení jeho dat. Ke směrování komunikace v ostatních případech byl pro účely testování vybrán modul Reactive Forwarding, který byl podrobně popsán v kapitole 4.3. Nicméně tento modul ve výchozím nastavení provádí směrování komunikace pouze na základě zdrojové a cílové MAC adresy. Vkládá tedy na SDN zařízení *flow* pravidla, která mají pouze jediné kritérium shody. Takto jednoduchá *flow* pravidla není možné namapovat na datové toky vytvořené modulem Traffic Analyzer. Tím pádem by nebylo možné pro vytvořené datové toky sbírat statistiky (např. počet paketů) ani sledovat přítomnost *flow* pravidla ve *flow* tabulce SDN zařízení. Z tohoto důvodu je modul Reactive Forwarding nutné nakonfigurovat tak, aby vkládal na SDN

zařízení *flow* pravidla, která obsahují následující kritéria shody: zdrojová a cílová IP adresa, zdrojové a cílové porty (UDP a TCP) a ICMP pole (typ a kód). Konfigurace se provádí v CLI ONOS kontroléru pomocí následujících příkazů[75] (uvedeny ve stejném pořadí jako příslušná kritéria shody):

```
cfg set org.onosproject.fwd.ReactiveForwarding matchIpv4Address true
cfg set org.onosproject.fwd.ReactiveForwarding matchTcpUdpPorts true
cfg set org.onosproject.fwd.ReactiveForwarding matchIcmpFields true
```

Ke generování datového provozu byly použity následující síťové utility:

- Ping[76] byl použit k vysílání ICMP zpráv typu *Echo Request* ve stanoveném intervalu.
- Netcat[77] byl použit k vysílání UDP a TCP segmentů obsahující data (např. textový řetězec), která si uživatel může stáhnout.
- Curl byl použit k testování rozpoznávání protokolů aplikační vrstvy (služeb), například HTTP, File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), telnet a další.

6.3 Uživatelská příručka

Jak již bylo uvedeno v úvodu do kapitoly 6, veškeré funkcionality modulu jsou dostupné skrze jeho GUI a monitorování datových toků je navíc podporováno také v CLI ONOS kontroléru. Oba uvedené typy uživatelských rozhraní jsou, stejně jako příslušná rozhraní ONOS kontroléru, kompletně v anglickém jazyce.

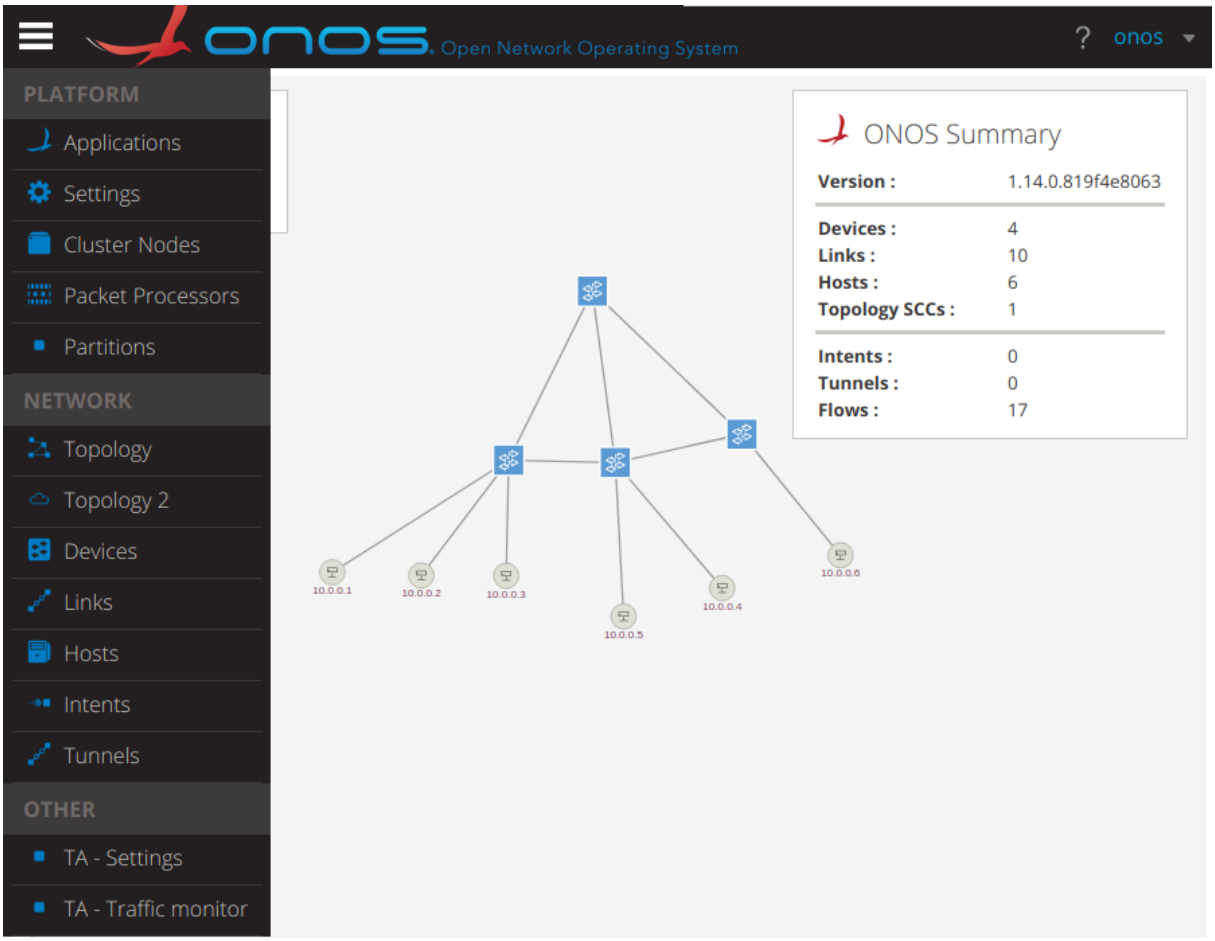
Tato podkapitola bude obsahovat stručný návod k obsluze modulu, ve kterém nebude zahrnuta jeho instalace a aktivace ani přístup do CLI a GUI ONOS kontroléru. Instalace a aktivace modulu byla popsána v kapitole 3.4 a přístup do GUI a CLI ONOS kontroléru v kapitole 3.3.

6.3.1 Přístup do GUI modulu

Po úspěšném přihlášení do GUI ONOS kontroléru je zobrazena jeho hlavní stránka. Po kliknutí na ikonu vlevo nahoře se zobrazí jeho navigační menu (znázorněno na obrázku 14) umožňující výběr pohledu ONOS aplikace, resp. modulu.

Modul Traffic Analyzer poskytuje dva pohledy, které lze nalézt v navigačním menu v kategorii „OTHER“ pod názvy „TA – Settings“ a „TA – Traffic monitor“ (vizte obrázek 14). Pohled Settings umožňuje nastavení modulu, zatímco Traffic monitor umožňuje monitoring a provádění operací s datovými toky a celou jejich kolekcí.

[75] První dva příkazy provádí zároveň konfiguraci kritérií shody dle zdrojové a cílové IP adresy, resp. zdrojového a cílového UDP/TCP portu.
[76] Ping umožňuje otestovat dosažitelnost koncového zařízení v síti, která používá IP protokol.
[77] Síťová utilita netcat umožňuje zápis a čtení z UDP a TCP soketů (Los, 2012).

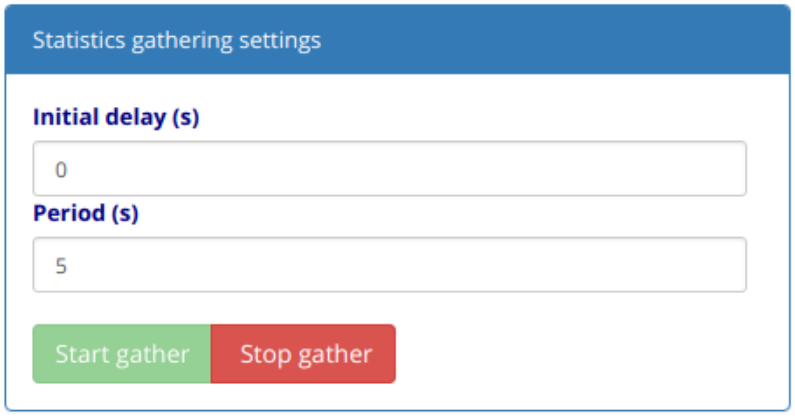


Obrázek 14: Navigační menu ONOS kontroléru

6.3.2 Nastavení modulu

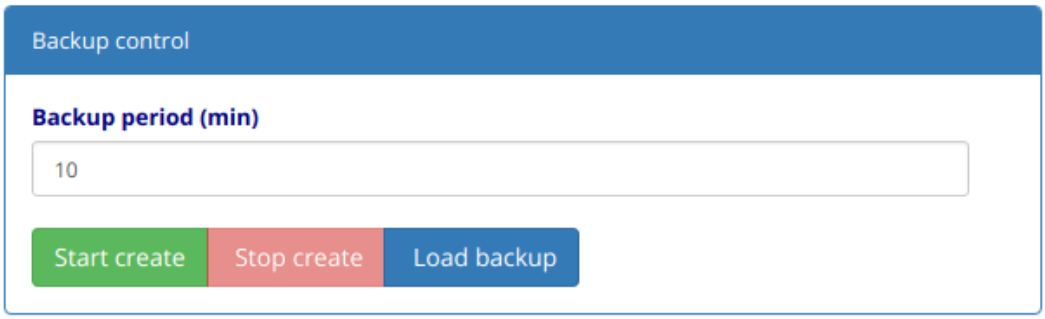
Nejprve je doporučeno provést nastavení modulu dle vlastních preferencí. Pohled Settings umožňující nastavení modulu se skládá ze tří na sobě nezávislých formulářů, které jsou znázorněny na obrázcích 15, 16 a 17. Tlačítka u jednotlivých formulářů se dynamicky zakazují, resp. povolují dle provedených akcí. Skutečnost, že tlačítko nelze použít, je indikována jednak snížením saturace jeho barvy a současně také ikonou kurzoru (přeškrtnutá kružnice). Při zadání hodnoty formuláře ve špatném formátu či jejím neuvedení je pod danou hodnotou zobrazeno upozornění, a navíc je zakázáno tlačítko umožňující provedení požadované akce.

Nastavení periodického sbírání statistik se provádí pomocí formuláře, který je znázorněn na obrázku 15. Formulář umožňuje kromě nastavení periody sbírání statistik také nastavení počátečního zpoždění, po jehož uplynutí jsou statistiky sbírány s nastavenou periodou. Nejmenší nastavitelnou periodou (a zároveň výchozím nastavením) je pět sekund, neboť v této periodě je aktualizována autoritativní tabulka, ze které modul sbírá statistiky.



Obrázek 15: Formulář umožňující sbírání statistik

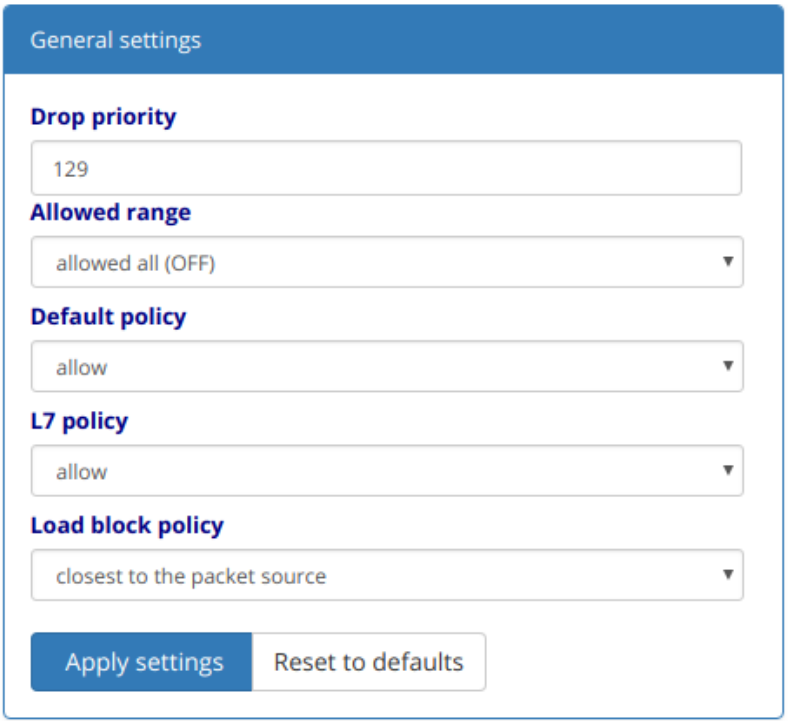
Na obrázku 16 je znázorněn formulář umožňující periodické vytváření a načítání záloh kolekce datových toků. Vytvořené zálohy jsou ukládány na serveru vždy pod stejným názvem na stejné umístění. Přičemž každá nová záloha provádí přepis té předchozí. Z tohoto důvodu není nutné při načítání její vyhledávání, stačí pouze stisknout tlačítko „Load backup“. Ve výchozím nastavení není periodické vytváření záloh zapnuté.



Obrázek 16: Formulář umožňující zálohování

Obecná nastavení modulu je možné provést pomocí formuláře, který je znázorněn na obrázku 17. Mezi obecná nastavení patří:

- priorita *flow* pravidel sloužících k blokování komunikace,
- povolený rozsah privátních IP adres (pakety mimo povolený rozsah jsou zahozeny),
- výchozí politika (zakázání, povolení, L7 inspekce) aplikovaná na pakety datových toků,
- výchozí politika L7 inspekce (povolení nebo zakázání komunikace v případě L7 inspekce)
- a výchozí politika blokování používaná při načítání kolekce datových toků ze souboru, která určuje, zdali bude pro blokovaný datový tok umístěno blokovací *flow* pravidlo nejblíže k jeho zdroji či cíli.

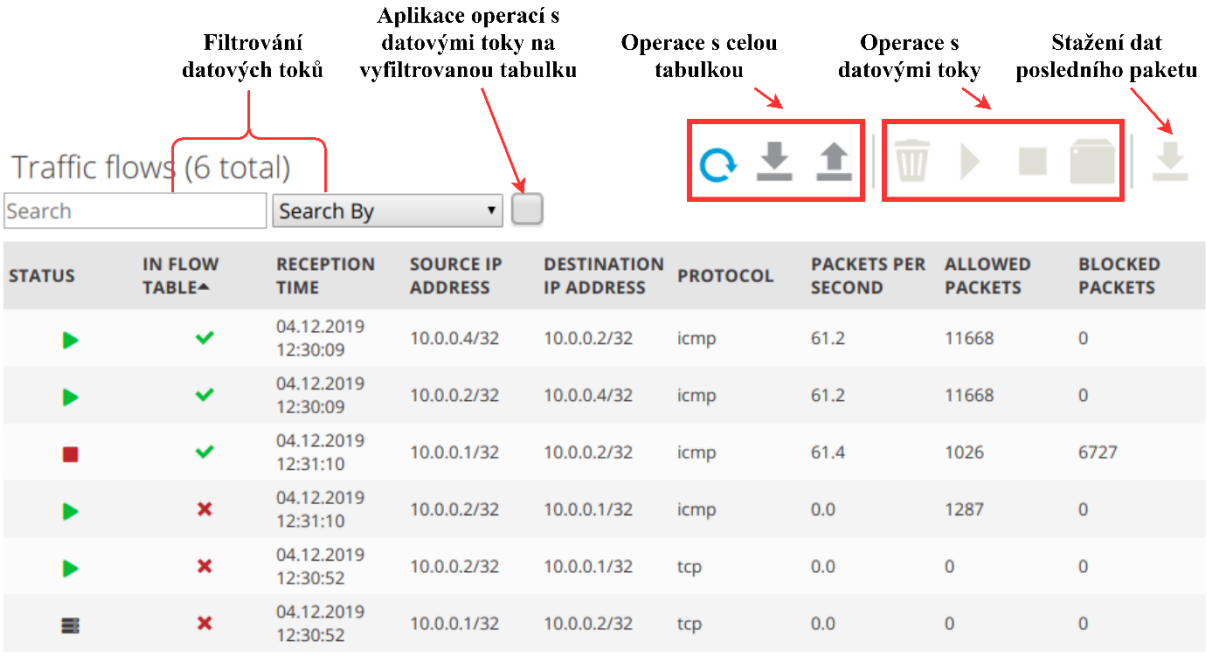


Obrázek 17: Formulář umožňující obecné nastavení modulu

6.3.3 Monitoring datových toků

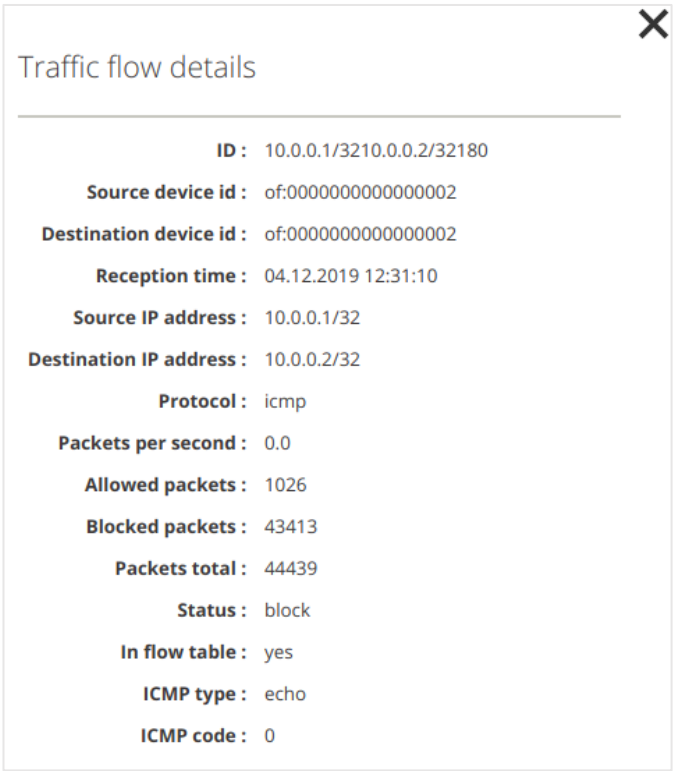
Monitorování datových toků je možné provádět pomocí tabulky (znázorněna na obrázku 18) v rámci pohledu Traffic monitor nebo alternativně v CLI ONOS kontroléru příkazem *print-traffic*. Oba uvedené přístupy umožňují filtrování a řazení datových toků podle zadaného kritéria. Nicméně příkaz *print-traffic* umožňuje, oproti pohledu, také filtrování pomocí více různých kritérií. Například je možné vyfiltrovat pouze datové toky s konkrétní zdrojovou IP adresou, které zároveň využívají ke komunikaci protokol TCP. Kompletní seznam nepovinných parametrů, včetně jejich popisu, lze vypsat zadáním následujícího příkazu v CLI ONOS kontroléru:

```
print-traffic --help
```



Obrázek 18: Pohled Traffic monitor

Na obrázku 19 je znázorněn dialog, který se automaticky zobrazí po kliknutí na konkrétní záznam v tabulce datových toků. Tento dialog zobrazuje další informace o konkrétním datovém toku, které nebylo možné do tabulky zahrnout, aniž by ztratila na přehlednosti.



Obrázek 19: Dialog zobrazující detaily datového toku

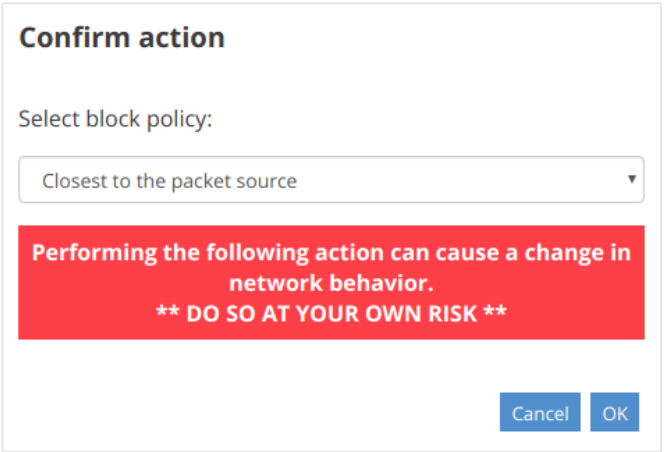
6.3.4 Operace s datovými toky

Kromě monitorování umožňuje pohled Traffic monitor také provádění operací s datovými toky. Operace je možné provádět pomocí tlačítek označených v obrázku 18 popiskem „Operace s datovými toky“. Tlačítka se opět dynamicky zakazují a povolují na základě stavu vybraného datového toku (např. povolený datový tok není možné povolit) a zaškrtávacího políčka označeného v obrázku 18 popiskem „Aplikace operací s datovými toky na vyfiltrovanou tabulku“. Pokud je vybrán konkrétní datový tok, provede se požadovaná operace pouze nad ním. Pokud je zaškrtnuto zaškrtávací políčko a zároveň není vybrán konkrétní datový tok, je operace provedena nad všemi datovými toky, které jsou v daný moment vyfiltrovány (zobrazeny v tabulce).

Modul Traffic Analyzer podporuje celkem čtyři operace s datovými toky (uvedeny v pořadí zleva doprava podle obrázku 18):

- Vymazání odstraní datový tok a příslušná *flow* pravidla.
- Povolení odstraní příslušné blokovací *flow* pravidlo datového toku – všechny jeho pakety jsou od této chvíle povoleny (jejich směrování zajišťuje jiný modul).
- Blokování vytvoří pro datový tok příslušné blokovací *flow* pravidlo – všechny jeho pakety jsou od této chvíle blokovány.
- L7 inspekce datového toku znamená, že všechny jeho pakety budou od této chvíle posílány na ONOS kontrolér, kde budou uložena jejich data. Po uložení jejich dat budou pakety buď směrovány k cíli modulem Traffic Analyzer nebo budou zahozeny (záleží na nastavení). L7 inspekci lze provádět pouze u TCP a UDP datových toků, neboť u ICMP paketů neumožňuje ONOS kontrolér získání dat.

Provedení každé výše uvedené operace je zapotřebí potvrdit v dialogu, který se zobrazí po stisknutí tlačítka umožňujícího provedení požadované operace. Jedinou výjimku představuje dialog (znázorněn na obrázku 20), sloužící kromě potvrzení blokovací operace také k výběru SDN zařízení, na které bude vloženo blokovací *flow* pravidlo (nejblíže ke zdroji či cíli).



Obrázek 20: Dialog sloužící k nastavení politiky blokování komunikace

Zvláštním typem operace je stažení souboru[78] obsahujícího data posledního zaznamenaného paketu datového toku (umístění tlačítka – vizte obrázek 18 vpravo nahoře). Tento typ operace, na rozdíl od výše uvedených, nemění stav datového toku a lze ho provádět, stejně jako L7 inspekci, pouze nad TCP a UDP datovými toky.

6.3.5 Operace s celou tabulkou

Poslední podporovanou funkcionalitou modulu jsou operace, které se týkají vždy celé kolekce datových toků (v GUI reprezentována tabulkou). Tyto operace je možné provádět v pohledu Traffic monitor pomocí tlačítek označených v obrázku 18 popiskem „Operace s celou tabulkou“. Pohled podporuje celkem tři operace (uvedeny v pořadí zleva doprava podle obrázku 18):

- Zapnutí/vypnutí automatického obnovení tabulky, která v rámci pohledu reprezentuje kolekci datových toků. Pokud je automatické obnovení vypnuto, nebude tabulka reflektovat současný stav kolekce datových toků. Pokud je automatické obnovení zapnuto, je tabulka obnovována každé dvě vteřiny.
- Stažení kolekce datových toků ve formátu JSON. Na tuto operaci se nevztahuje filtrování datových toků (vizte obrázek 18), stažený soubor vždy obsahuje kompletní kolekci.
- Nahrání kolekce datových toků ve formátu JSON a její následné mapování na topologii. Pokud je nahrávaný soubor v jiném formátu či poškozen, je zobrazeno příslušné upozornění ve formě dialogu (v případě špatného formátu) či zprávy (v případě poškozeného souboru).

[78] Binární data paketu jsou zakódována do sekvence tisknutelných znaků pomocí Base64 kódování.

ZÁVĚR

V teoretické části práce byl ONOS kontrolér popsán tak, aby čtenář získal ucelené znalosti týkající se zejména vývoje modulu pro tento kontrolér. Tyto znalosti je totiž z oficiální dokumentace ONOS kontroléru, která je zastaralá a nekompletní, velmi těžké získat. Popis ONOS kontroléru se nicméně neopírá pouze o citované materiály, ale také o praktické zkušenosti nabyté při práci s ONOS kontrolérem a při vývoji vlastního modulu. Díky tomu je možné čtenáři poskytnout konzistentní informace a ověřené postupy, které lze okamžitě aplikovat v praxi.

Při vytváření modulu Traffic Analyzer jsem se naučil pracovat s frameworkem AngularJS a OSGi. Největší překážkou při vytváření modulu byl vývoj GUI, resp. jeho integrace do GUI ONOS kontroléru. Při vývoji GUI modulu bylo zapotřebí celou vygenerovanou strukturu upravit tak, aby bylo možné GUI snadno rozšířit o další pohledy bez duplicit zdrojového kódu – při dodržení osvědčených principů návrhu softwaru (konkrétně principu *Don't Repeat Yourself*). Samotná úprava vygenerované struktury není nikde popsána, a proto její provedení bylo velmi obtížné. Dalším problémem bylo, že ONOS kontrolér využívá zastaralou verzi frameworku AngularJS (1.3.5), pro kterou je mnohem složitější vyhledat řešení vyvstanuvších problémů než u aktuální verze. Rovněž bylo také obtížné vytváření sofistikovanějších dialogových oken, která umožňují například výběr z více možností (HTML element *select*). V oficiální dokumentaci ONOS kontroléru je uvedena pouze metoda umožňující vytvoření blokového HTML elementu *div*. Konkrétní javascriptovou knihovnu *d3.js*, kterou GUI ONOS kontroléru využívá k vytváření HTML elementů, bylo zapotřebí dohledat ve zdrojovém kódu ONOS kontroléru. Poslední překážkou bylo řešení vyvstanuvších problémů v rámci implementace business funkcionalit modulu. Z důvodu nedostatku informací bylo jejich řešení časově velmi náročné.

Modul Traffic Analyzer poskytuje všechny funkcionality, které byly stanoveny v zadání. Navíc byla implementována L7 inspekce datových toků. Modul lze rozšířit o další funkcionality, např. vlastní směrování komunikace a podpora IPv6. V současném stavu je možné modul použít v praxi, nicméně pouze za předpokladu, že spravovaná topologie používá protokol IPv4 a že je ke směrování komunikace použit správně nakonfigurovaný modul Reactive Forwarding. Klíčové komponenty modulu jsou navíc použity v rámci projektu pro Technologickou agenturu České republiky (ZÉTA – Program na podporu aplikovaného výzkumu), jehož cílem je vývoj inteligentního firewallu sloužícího k zabezpečení průmyslových sítí.

POUŽITÁ LITERATURA

About The Linux Foundation, c2019. *The Linux Foundation* [online]. San Francisco: The Linux Foundation® [cit. 2019-01-21]. Dostupné z: https://www.linuxfoundation.org/about/.

ADARA NETWORKS, 2016. Rabbit MQ Tutorial. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-28]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Rabbit+MQ+Tutorial#RabbitMQTutorial-MQSupportedONOSeventtypes./.

Architecture – OSGi™ Alliance, c2019. *OSGi™ Alliance – The Dynamic Module System for Java* [online]. OSGi™ Alliance [cit. 2019-02-05]. Dostupné z: https://www.osgi.org/developer/architecture/.

Bazel – a fast, scalable, multi-language and extensible build system", c2019. *Bazel* [online]. Google [cit. 2019-03-09]. Dostupné z: https://bazel.build/.

BERDE, Pankaj et al., 2014. ONOS: Towards an Open, Distributed SDN OS. *Proceedings of the third workshop on Hot topics in software defined networking – HotSDN '14* [online]. New York, New York, USA: ACM Press, 1-6 [cit. 2019-01-23]. DOI: 10.1145/2620728.2620744. ISBN 9781450329897. Dostupné z: http://dl.acm.org/citation.cfm?doid=2620728.2620744.

BERNERS-LEE, Timothy et al., 2005. RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax. *IETF Tools* [online]. Internet Research Task Force [cit. 2019-01-22]. Dostupné z: https://tools.ietf.org/html/rfc3986.

Black Hole, c2019. *Techopedia – Where Information Technology and Business Meet* [online]. Techopedia [cit. 2019-02-07]. Dostupné z: https://www.techopedia.com/definition/3159/black-hole.

BOSWELL, David, 2017. GRPC brigade. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/gRPC+brigade.

BOSWELL, David a Carmelo CASCONE, 2019. P4 brigade. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-28]. Dostupné z: https://wiki.onosproject.org/display/ONOS/P4+brigade.

CARL-MITCHELL, Smoot a John S. QUARTERMAN, 1987. RFC 1027 - Using ARP to implement transparent subnet gateways. *IETF Tools* [online]. Internet Research Task Force [cit. 2019-01-27]. Dostupné z: https://tools.ietf.org/html/rfc1027.

CASCONE, Carmelo, 2018. P4Runtime support in ONOS. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-28]. Dostupné z: https://wiki.onosproject.org/display/ONOS/P4Runtime+support+in+ONOS.

COKER, Oswald a Siamak AZODOLMOLKY, 2017. *Software-Defined Networking with OpenFlow*. Second Edition. Birmingham: Packt Publishing. ISBN 978-1-78398-428-2.

CONDON, Sean, 2017. Layer 2 Monitoring with CFM and Services OAM. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Layer+2+Monitoring+with+CFM+and+Services+OAM.

CORD Platform | Central Office Rearchitected as a Datacenter | ONF, c2019. *Open Networking Foundation is an operator ...*[online]. Open Networking Foundation [cit. 2019-01-30]. Dostupné z: https://www.opennetworking.org/cord/.

Curl, [1997]. *Curl* [online]. MIT/X [cit. 2019-03-09]. Dostupné z: https://curl.haxx.se/.

DE LEENHEER, Marc a Andrea CAMPANELLA, 2017. TL1. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-30]. Dostupné z: https://wiki.onosproject.org/display/ONOS/TL1.

Differences between JDK, JRE and JVM, [2015]. *GeeksforGeeks | A computer science portal for geeks* [online]. geeksforgeeks [cit. 2019-03-09]. Dostupné z: https://www.geeksforgeeks.org/differences-jdk-jre-jvm/.

EDDY, Rusty, 2015. ONOS Multicast Forwarding Architecture. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/ONOS+Multicast+Forwarding+Architecture.

ESTRIN, Deborah et al., 1997. RFC 2117 - Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. *IETF Tools* [online]. Internet Research Task Force [cit. 2019-01-27]. Dostupné z: https://tools.ietf.org/html/rfc2117.

Java 8 Predicate with Examples, 2018. *GeeksforGeeks | A computer science portal for geeks* [online]. geeksforgeeks [cit. 2019-04-11]. Dostupné z: https://www.geeksforgeeks.org/java-8-predicate-with-examples/.

GORANSSON, Paul, Chuck BLACK a Timothy CULVER, c2017. *Software Defined Networks: a comprehensive approach*. Second edition. Singapore: Morgan Kaufmann. ISBN 978-0-12-804555-8.

HART, Jonathan, 2016a. Flow Rule Subsystem. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-02-07]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Flow+Rule+Subsystem.

HART, Jonathan, 2016b. Neighbour Resolution Service. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Neighbour+Resolution+Service.

Headend, c2019. *Techopedia – Where Information Technology and Business Meet* [online]. Techopedia [cit. 2019-01-30]. Dostupné z: https://www.techopedia.com/definition/7550/headend.

HOHN, Alan, 2016. Apache Karaf Features for OSGi Deployment. *DZone* [online]. [cit. 2019-02-05]. Dostupné z: https://dzone.com/articles/apache-karaf-features-for-osgi-deployment.

HONG, Sungmin et al., 2015. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures. *Proceedings 2015 Network and Distributed System Security Symposium* [online]. Reston, VA: Internet Society, 2015, - [cit. 2019-01-31]. DOI: 10.14722/ndss.2015.23283. ISBN 1-891562-38-X. Dostupné z: https://www.ndss-symposium.org/ndss2015/ndss-2015-programme/poisoning-network-visibility-software-defined-networks-new-attacks-and-countermeasures/.

IEEE 802.1ag-2007 - IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks Amendment 5: Connectivity Fault Management, 2007. *IEEE Standards Association logo* [online]. IEEE [cit. 2019-01-27]. Dostupné z: https://standards.ieee.org/standard/802_1ag-2007.html.

Internet Exchange Point (IXP), c2019. *Techopedia – Where Information Technology and Business Meet* [online]. Techopedia [cit. 2019-01-27]. Dostupné z: https://www.techopedia.com/definition/27705/internet-exchange-point-ixp.

Introduction to Archetypes, c2002-2019. *Welcome to The Apache Software Foundation!* [online]. The Apache Software Foundation [cit. 2019-03-17]. Dostupné z: https://maven.apache.org/guides/introduction/introduction-to-archetypes.html.

ISMAIL, Kaya, 2018. What Is a Single Page Application?. *Covering Digital Experience, Digital Workplace, Information Management* [online]. Simpler Media Group [cit. 2019-03-13]. Dostupné z: https://www.cmswire.com/digital-experience/what-is-a-single-page-application/.

JAMPANI, Madan a Mao JIANWEI, 2016. Network Topology State. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-23]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Network+Topology+State.

Java Runtime Environment (JRE), c2019. *Techopedia – Where Information Technology and Business Meet* [online]. Techopedia [cit. 2019-03-09]. Dostupné z: https://www.techopedia.com/definition/5442/java-runtime-environment-jre.

KOSHIBE, Ayaka, 2016. Packet Optical Convergence. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Packet+Optical+Convergence.

KOSHIBE, Ayaka a Bob LANTZ, 2017. Developer Guide. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-03-09]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Developer+Guide.

KOSHIBE, Ayaka a Elena OLKHOVSKAYA, 2016. Cluster Coordination. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-23]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Cluster+Coordination.

KOSHIBE, Ayaka a Jonathan HART, 2016a. Intent Framework. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-02-05]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Intent+Framework.

KOSHIBE, Ayaka a Jonathan HART, 2016b. SDN-IP Architecture. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/SDN-IP+Architecture.

KOSHIBE, Ayaka a Mao JIANWEI, 2018. Installing and running ONOS. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-03-09]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Installing+and+running+ONOS.

KOSHIBE, Ayaka a Muhammad SHAKIL, 2017. Device Subsystem. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-02-01]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Device+Subsystem.

KOSHIBE, Ayaka a Luca PRETE, 2017. The ONOS CLI. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-02-06]. Dostupné z: https://wiki.onosproject.org/display/ONOS/The+ONOS+CLI.

KOSHIBE, Ayaka a You WANG, 2016. System Components. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-22]. Dostupné z: https://wiki.onosproject.org/display/ONOS/System+Components.

KOSHIBE, Ayaka et al., 2016. Network State Construction. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-31]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Network+State+Construction.

KRUSE, Megan, 2018. What is BGP Hijacking, Anyway?. *Internet Society* [online]. USA: Internet Society [cit. 2019-01-27]. Dostupné z: https://www.internetsociety.org/blog/2018/05/what-is-bgp-hijacking-anyway/.

KUMAR, Himal, 2017. Castor Project. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Castor+Project.

LIN, Pingping, 2016. SDN-IP Reactive Routing. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/SDN-IP+Reactive+Routing.

LIN, Pingping a Jonathan HART, 2015. Virtual BNG. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Virtual+BNG.

LOS, Petr, 2012. Netcat – Švýcarský armádní nůž pro TCP/IP. *AbcLinuxu.cz - Linux na stříbrném podnose* [online]. Nitemedia [cit. 2019-04-14]. Dostupné z: http://www.abclinuxu.cz/clanky/netcat-svycarsky-armadni-nuz-pro-tcp-ip.

MAVROMMATIS, Dimitris, 2018. ARTEMIS: an Automated System against BGP Prefix Hijacking. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-27]. Dostupné z: https://wiki.onosproject.org/display/ONOS/ARTEMIS%3A+an+Automated+System+against+BGP+Prefix+Hijacking.

MOKRÁČEK, Jan, 2017. *Analýza kontrolérů softwarově definovaných sítí*. Pardubice. Bakalářská práce. Univerzita Pardubice, Fakulta elektrotechniky a informatiky. Vedoucí práce Filip Holík.

MOON, Hyunsun a Jian LI, 2018. SONA: DC Network Virtualization. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-28]. Dostupné z: https://wiki.onosproject.org/display/ONOS/SONA%3A+DC+Network+Virtualization.

MORO, Daniele a Davide SANVITO, 2017. IMR – Intent Monitor and Reroute service. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-28]. Dostupné z: https://wiki.onosproject.org/display/ONOS/IMR+-+Intent+Monitor+and+Reroute+service.

MURRELL, Will, c2000–2019. What does the Intent Framework do in the ONOS SDN platform?. *TechTarget* [online]. TechTarget [cit. 2019-02-05]. Dostupné z: https://searchnetworking.techtarget.com/answer/What-does-the-Intent-Framework-do-in-the-ONOS-SDN-platform.

NADEAU, Thomas D. a Kenneth GRAY, 2013. *SDN: software defined networks*. Sebastopol: O´Reilly Media. ISBN 978-1-449-34230-2.

Network Operators, c2019. *Cloud-Optimized Real-Time Communications Solutions | Dialogic* [online]. Dialogic Corporation [cit. 2019-01-30]. Dostupné z: https://www.dialogic.com/glossary/network-operators.

Neutron/ML2, [201?]. *OpenStack* [online]. [cit. 2019-01-28]. Dostupné z: https://wiki.openstack.org/wiki/Neutron/ML2.

OKTIAN et al., 2017. Distributed SDN controller system: A survey on design choice. *Computer Networks* [online]. 121, 100-111 [cit. 2019-01-23]. DOI: 10.1016/j.comnet.2017.04.038. ISSN 13891286. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S1389128617301706.

Open and Disaggregated Transport Network (ODTN), c2019. *Open Networking Foundation is an operator ...* [online]. Open Networking Foundation [cit. 2019-01-28]. Dostupné z: https://www.opennetworking.org/odtn/.

OpenFlow Switch Specification: Version 1.5.1 (Protocol version 0x06), 2015. *Open Networking Foundation is an operator ...*[online]. Open Networking Foundation [cit. 2019-01-23]. Dostupné z: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

Open Networking Foundation and ON.Lab to Merge to Accelerate Adoption of SDN, c2019. *Open Networking Foundation is an operator ...* [online]. Open Networking Foundation [cit. 2019-01-20]. Dostupné z: https://www.opennetworking.org/news-and-events/press-releases/open-networking-foundation-and-on-lab-to-merge-to-accelerate-adoption-of-sdn/.

POM Reference, c2002-2019. *Welcome to The Apache Software Foundation!* [online]. The Apache Software Foundation [cit. 2019-03-19]. Dostupné z: https://maven.apache.org/pom.html#The_Basics.

RADOSLAVOV, Pavlin, 2015. Experimental Features. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-01-28]. Dostupné z: https://wiki.onosproject.org/display/ONOS11/Experimental+Features.

RAO, Sridhar, 2015. SDN Series Part Seven: ONOS. *The New Stack* [online]. The New Stack, 2015 [cit. 2019-01-20]. Dostupné z: https://thenewstack.io/open-source-sdn-controllers-part-vii-onos/.

ROBINSON, Scott, 2013. What is Maven?. *Stack Abuse* [online]. Stack Abuse [cit. 2019-02-05]. Dostupné z: https://stackabuse.com/what-is-maven/.

ROUSE, Margaret, 2011a. Policy-based management. *TechTarget* [online]. TechTarget [cit. 2019-02-05]. Dostupné z: https://whatis.techtarget.com/definition/policy-based-management.

ROUSE, Margaret, 2011b. User self-provisioning. *TechTarget* [online]. TechTarget [cit. 2019-01-27]. Dostupné z: https://searchitchannel.techtarget.com/definition/user-self-provisioning.

ROUSE, Margaret, 2007. Policy-based networking. *TechTarget* [online]. TechTarget [cit. 2019-01-21]. Dostupné z: https://searchnetworking.techtarget.com/definition/policy-based-networking.

ROUSE, Margaret, 2014a. Mission-critical application. *TechTarget* [online]. TechTarget [cit. 2019-01-20]. Dostupné z: https://searchitoperations.techtarget.com/definition/mission-critical-computing.

ROUSE, Margaret, 2014b. Domain specific language (DSL). *TechTarget* [online]. TechTarget [cit. 2019-01-28]. Dostupné z: https://searchmicroservices.techtarget.com/definition/domain-specific-language-DSL.

ROUSE, Margaret a Ivy WIGMORE, 2014. Eventual consistency. *TechTarget* [online]. TechTarget [cit. 2019-02-06]. Dostupné z: https://whatis.techtarget.com/definition/eventual-consistency.

ROUSE, Margaret a Ivy WIGMORE, 2016. Monolithic architecture. *TechTarget* [online]. TechTarget [cit. 2019-02-04]. Dostupné z: https://whatis.techtarget.com/definition/monolithic-architecture.

ROUSE, Margaret, 2017. Deep packet inspection (DPI). *TechTarget* [online]. TechTarget [cit. 2019-04-05]. Dostupné z: https://searchnetworking.techtarget.com/definition/deep-packet-inspection-DPI.

SANVITO, Davide et al., 2018. ONOS Intent Monitor and Reroute service. *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)* [online]. IEEE, 2018, 272-276 [cit. 2019-01-28]. DOI: 10.1109/NETSOFT.2018.8460064. ISBN 978-1-5386-4633-5. Dostupné z: https://ieeexplore.ieee.org/document/8460064/.

SUBRAMANIAN, Sriram a Sreenivas VORUGANTI, 2016. *Software-Defined Networking (SDN) with OpenStack*. Birmingham – Mumbai: Packt Publishing. ISBN 978-1-78646-599-3.

SWANSON, Alain, 2016. Tenant networks vs. provider networks in the private cloud context. *Superuser, an Online Publication Covering Open Infrastructure* [online]. Superuser [cit. 2019-01-28]. Dostupné z: http://superuser.openstack.org/articles/tenant-networks-vs-provider-networks-in-the-private-cloud-context/.

THANGAM, Sivaram Vargheese, 2015. OSGI COMPONENT VS SERVICE IN AEM. *Compute Patterns* [online]. [cit. 2019-03-22]. Dostupné z: http://www.computepatterns.com/76/osgi-component-vs-service-in-aem/.

Transactional Language 1: Beginner's Guide To This Telemetry Protocol, [198?]. *DPS Telecom – Remote Monitoring and Control Equipment* [online]. DPS Telecom [cit. 2019-01-30]. Dostupné z: https://www.dpstele.com/network-monitoring/alarm/tl1/what-is.php.

Transport API (TAPI) 2.0 Overview, 2017. *Open Networking Foundation is an operator ...* [online]. Open Networking Foundation [cit. 2019-01-30]. Dostupné z: https://www.opennetworking.org/wp-content/uploads/2017/08/TAPI-2-WP_DRAFT.pdf.

TYSON, Matthew, 2018. What is the JVM? Introducing the Java virtual machine. *Welcome to JavaWorld.com* [online]. IDG Communications [cit. 2019-02-05]. Dostupné z: https://www.javaworld.com/article/3272244/core-java/what-is-the-jvm-introducing-the-java-virtual-machine.html.

Uniform Resource Locator (URL), c2019. *Techopedia – Where Information Technology and Business Meet* [online]. Techopedia [cit. 2019-03-10]. Dostupné z: https://www.techopedia.com/definition/1352/uniform-resource-locator-url.

Using the console, c2018. *Welcome to The Apache Software Foundation!* [online]. [cit. 2019-03-23]. Dostupné z: https://svn.apache.org/repos/asf/karaf/site/production/manual/latest-3.0.x/console.html.

VACHUSKA, Thomas, 2015. Application Subsystem. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-02-06]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Application+Subsystem.

VACHUSKA, Thomas a Ayaka KOSHIBE, 2016. Configuring Authentication – CLI, GUI & REST API. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-03-10]. Dostupné z: https://wiki.onosproject.org/pages/viewpage.action?pageId=4162614.

VACHUSKA, Thomas a Markus GAUGUSCH, 2016. Creating and deploying an ONOS application. *Wiki Home – ONOS – Wiki*[online]. [cit. 2019-02-06]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Creating+and+deploying+an+ONOS+application.

VACHUSKA, Thomas a Pradeep Reddy KANDI, 2016. Device Driver Subsystem. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-02-04]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Device+Driver+Subsystem.

VACHUSKA, Thomas a Yuta HIGUCHI, 2016. Component Configuration. *Wiki Home – ONOS – Wiki* [online]. [cit. 2019-02-06]. Dostupné z: https://wiki.onosproject.org/display/ONOS/Component+Configuration.

VALKOVIČ, Patrik, c2019. Immutable objects (neměnné objekty). *Itnetwork.cz - Ajťácká sociální síť a materiálová základna pro C#, Java, PHP, HTML, CSS, JavaScript a další.* [online]. itnetwork.cz [cit. 2019-02-04]. Dostupné z: https://www.itnetwork.cz/navrh/navrhove-vzory/nemenne-objekty-immutable-objects.

WEISSBERGER, Alan, 2019. IHS Markit: SDN deployed by 78 % of global service providers at end of 2018. *ComSoc Technology Blog* [online]. IEEE Communications Society [cit. 2019-04-18]. Dostupné z: http://techblog.comsoc.org/2019/01/28/ihs-markit-sdn-deployed-by-78-of-global-service-providers-at-end-of-2018/.

What is gRPC?, c2019. *A high-performance, open-source universal RPC framework* [online]. The gRPC Authors [cit. 2019-01-27]. Dostupné z: https://grpc.io/docs/guides/.

What is a mobile-first approach? Definition, c2018. *Analytics Suite 2, AT Internet Web Analytics Solution* [online]. AT INTERNET [cit. 2019-04-17]. Dostupné z: https://www.atinternet.com/en/glossary/mobile-first/.

What is OpenStack?, [201?]. *OpenStack* [online]. [cit. 2019-01-28]. Dostupné z: https://www.openstack.org/software/.

What is peering?, c2019. *Netnod* [online]. Netnod [cit. 2019-03-06]. Dostupné z: https://www.netnod.se/ix/what-is-peering.

PŘÍLOHY

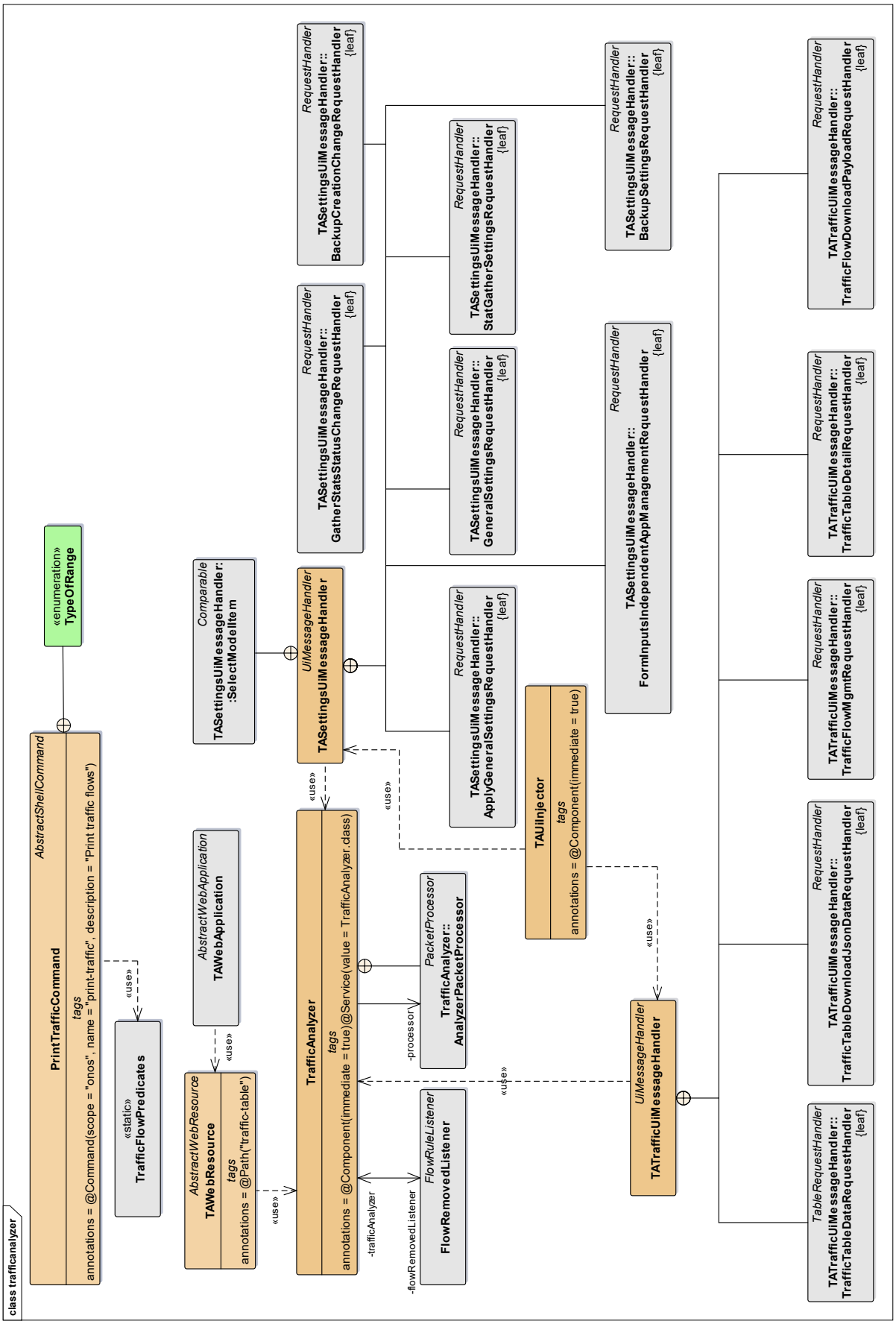
Příloha A – Vlastnosti modulu v souboru pom.xml

```
<properties>
     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
     <onos.version>2.0.0</onos.version>
     <!-- Uncomment to generate ONOS app from this module.
     <api.version>1.0.0</api.version>
     <onos.app.name>org.foo.app</onos.app.name>
     <onos.app.title>Foo App</onos.app.title>
     <onos.app.origin>Foo, Inc.</onos.app.origin>
     <onos.app.category>default</onos.app.category>
     <api.description>Sample application REST API</api.description>
     <web.context>/onos/foo-app</web.context>
     <api.title>Sample app REST API</api.title>
     <onos.app.url>http://onosproject.org</onos.app.url>
     <api.package>org.foo.app</api.package>
     <onos.app.readme>ONOS OSGi bundle archetype.</onos.app.readme>
     <onos.app.requires>org.onosproject.fwd,…</onos.app.requires>
     -->
</properties>
```

Příloha B – Zdrojový kód souboru AppUiComponent.java

```
1. @Component(immediate = true)
2. public class AppUiComponent {
3.
4.   private static final String VIEW1_ID = "viewId1";
5.   private static final String VIEW1_TEXT = "view1_text";
6.
7.   private static final String VIEW2_ID = "viewId2";
8.   private static final String VIEW2_TEXT = "view2_text";
9.
10.  private final Logger log = LoggerFactory.getLogger(getClass());
11.
12.  @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
13.  protected UiExtensionService uiExtensionService;
14.
15.  private final List<UiView> uiViews = ImmutableList.of(
16.       new UiView(UiView.Category.OTHER, VIEW1_ID, VIEW1_TEXT),
17.       new UiView(UiView.Category.OTHER, VIEW2_ID, VIEW2_TEXT));
18.
19.  private final UiMessageHandlerFactory messageHandlerFactory =
20.       () -> ImmutableList.of(
21.                 new AppUiView1MessageHandler(),
22.                 new AppUiView2MessageHandler()
23.       );
24.
25.  protected UiExtension extension =
26.       new UiExtension.Builder(getClass().getClassLoader(), uiViews)
27.                 .resourcePath("glueFiles")
28.                 .messageHandlerFactory(messageHandlerFactory)
29.                 .build();
30.  @Activate
31.  protected void activate() {
32.       uiExtensionService.register(extension);
33.       log.info("UIS injected");
34.  }
35.
36.  @Deactivate
37.  protected void deactivate() {
38.       uiExtensionService.unregister(extension);
39.       log.info("UIS deactivated");
40.  }
41.
42. }
```

Příloha C – UML diagram tříd balíčku trafficanalyzer



Příloha D – UML diagram tříd zbývajících balíčků modulu