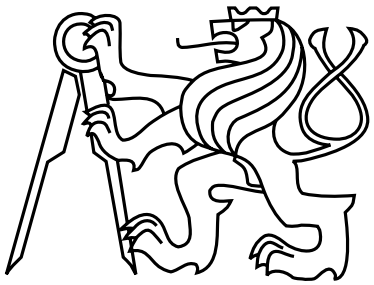
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra telekomunikační techniky



Diplomová práce

Automatizovaný systém konfigurace SDN síťových zařízeních

Bc. František Flachs

Vedoucí práce: doc. Ing. Leoš Boháč, Ph.D.

Studijní program: Elektronika a komunikace, Magisterský

Obor: Komunikační sítě a internet

květen 2020

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 22. 5. 2020 .....

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Flachs** Jméno: **František** Osobní číslo: **420080**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Komunikační sítě a internet**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Automatizovaný systém konfigurace SDN síťových zařízeních

Název diplomové práce anglicky:

Automated system configuration of SDN network devices

Pokyny pro vypracování:

Navrhněte systém pro automatizované zavedení nástrojů k výuce v laboratoři počítačových sítí. Analyzujte požadavky na hardware i software a omezení plynoucí z vlastností síťových zařízení.
Proveďte průzkum potřeb práce studentů v laboratoři a přizpůsobte systém pro co nejjednodušší a nejefektivnější použití. Do průzkumu zahrňte rovněž potřeby vyučujících pro snadnou přípravu laboratoře k výuce daného tématu. Vezměte v potaz čas nutný pro zavedení konfigurací a zprovoznění zařízení na začátku výuky.

Seznam doporučené literatury:

[1] EDELMAN, Jason, Scott LOWE a Matt OSWALT. Network programmability and automation: skills for the next-generation network engineer. Sebastopol, California: O'Reilly Media, 2018. ISBN 978-1-49193-125-7.
[2] GORANSSON, Paul, Chuck BLACK a Timothy CULVER. Software defined networks: a comprehensive approach. Second edition. Singapore: Morgan Kaufmann, [2017]. ISBN 0-12804-555-8.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Leoš Boháč, Ph.D., katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **08.01.2020** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2021**

doc. Ing. Leoš Boháč, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Summary

This thesis deals with the design and the implementation of an automated system for network device configuration. This system consists of several main components, such as interfaces for communicating with SDN controllers, interfaces for communicating with a virtual network, and a set of network device configuration tests, including a tool that performs them. This automated system is based on a model-view-controller software architecture. Scalability is provided by adding instances of SDN controllers that can be linked to a virtual network. The system is designed for network education academies to teach software defined networks. For learning purposes the system is designed to automatically run the required network controller, virtual network with defined parameters, including the OpenFlow protocol version, and to run a set of defined tests on a selected topology. A student receives feedback to correct or change the configuration after running the tests.

Index terms: SDN, network virtualization, automation, networking academy, Python, Mininet, OpenFlow, ONOS, OpenDaylight, Floodlight, POX, Ryu

Anotace

Tato diplomová práce se zabývá návrhem a implementací automatizovaného systému pro konfiguraci síťových zařízeních. Systém se skládá z několika hlavních komponent, kterými jsou rozhraní pro komunikaci s SDN kontroléry, rozhraní pro komunikaci s virtuální sítí a sada testů konfigurací síťových zařízeních včetně nástroje, který testy provádí. Automatizovaný systém je postaven na softwarové architektuře model-view-controller. Škálovatelnost je zajištěna prostřednictvím instancí SDN kontrolérů, které je možné následně propojit s virtuální sítí. Systém je navržen pro síťové vzdělávací akademie k výuce softwarově definovaných sítí. Pro potřeby výuky je systém navržen tak, aby automatizovaně spustil požadovaný kontrolér sítě, virtuální síť s definovanými parametry včetně verze OpenFlow protokolu a na vybrané topologii umožnil spouštět sadu definovaných testů. Student po spuštění testů získá zpětnou vazbu, na základě které může následně konfiguraci opravit či změnit.

Klíčová slova: SDN, virtualizace sítě, automatizace, síťová akademie, Python, Mininet, OpenFlow, ONOS, OpenDaylight, Floodlight, POX, Ryu

Poděkování

Rád bych poděkoval vedoucímu své diplomové práce doc. Ing. Leoši Boháčovi, Ph.D. za inspirativní i technické rady a za jeho ochotu, kterou mi věnoval. Mé poděkování patří i kolegům ze společnosti Networksys a.s. za cenné rady, konzultace a poskytnuté know-how při realizaci této práce. V neposlední řadě bych chtěl také poděkovat své rodině a blízkým přátelům za pomoc i trpělivost při mém studiu.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Kapitola 1

Úvod

V diplomové práci se zabývám popisem moderních datových sítí, které jsou označované jako softwarově definované sítě. Vlivem rychle rostoucích požadavků na IT infrastrukturu a dynamickou konfiguraci síťových zařízeních, není tradiční pojetí datových sítí mnohdy plně dostačující. Sítě tohoto typu mají nedostatky při nasazování nových síťových služeb a z pohledu lidských zdrojů není vždy možné zajistit včas a správně veškeré požadavky na nastavení sítě různých organizací.

V softwarově definovaných sítích mají síťová zařízení oddělenou datovou část od řídicí části a veškerá logická funkcionalita, rozhodovací proces a řízení je přesunuto do řídicího prvku sítě, kterým je SDN kontrolér. Aby se předešlo vzniku *single point of failure*, je v síti přítomno zpravidla několik kontrolérů, které spolu komunikují a synchronizují na pozadí své databáze o síťových zařízeních.

1.1 Motivace

Motivací pro napsání této diplomové práce je umožnění snadného začátku výuky softwarově definovaných sítí na školách a v organizacích provozujících síťové akademie. V současné době, pokud na některé technické škole probíhá výuka, tak zřejmě doposud nebyly tematické plány připraveny tak, aby se dostatečně věnovaly výuce této problematiky.

Navržený systém automatizovaně spouští vybranou topologii, tudíž student, který ho používá, nemusí nezbytně nutně stahovat různé nástroje z internetu a trávit několik hodin až dnů samotnou přípravou prostředí, se kterým následně bude pracovat.

Veškerá práce a propojené komponenty jsou distribuovány ve virtuálním prostředí, které je možné spustit na dnes běžně používaných virtualizačních platformách. Tento způsob distribuce umožňuje zejména snadno vytvářet *snapshoty*, které umožní studentovi vrátit se do požadovaného předchozího stavu.

Současně lektoři, kteří mají na starost přípravu výuky, nemusí trávit drahocenný čas přípravou topologií a konfigurací jednotlivých SDN kontrolérů pro prvotní spuštění, protože příprava systému je provedena automaticky.

Velkou výhodou tohoto systému je úspora času nejen lektorů, ale i studentů. Lze téměř okamžitě začít s konfigurací připravených zadání a seznamováním se s technologií SDN.

1.2 SDN kontrolér

Hlavní komponentou softwarově definované sítě je kontrolér. Základní funkcionalitou kontroléru je efektivně řídit topologii sítě, komunikovat s jednotlivými síťovými prvky, realizovat jejich programování a sbírat data ze síťových zařízeních za účelem analýzy toků dat v konkrétním úseku sítě. Nově vzniká centralizované řízení sítě oproti staršímu, distribuovanému. Díky použití SDN kontroléru lze snadno a rychle nasadit nová pravidla pro datové toky napříč celou spravovanou infrastrukturou.

Typickým protokolem pro komunikaci kontroléru se síťovým zařízením je protokol OpenFlow, pomocí kterého lze ovlivňovat způsob směrování či přepínání toků dat v síti. Při selhání kontroléru hrozí selhání celé sítě, a proto bývá provozován v clusteru nebo na virtuálním stroji s vysokou dostupností. Síť je schopna směrovat provoz i při dočasném výpadku kontroléru, a to do doby expirace OpenFlow záznamu na síťovém zařízení.

V této práci je použito několik SDN kontrolérů, které jsou implementovány pro použití studenty při konfiguraci jednotlivých zadáních. Každý použitý kontrolér je ve svém ohledu specifický, ať už je to přítomnost vlastního grafického rozhraní, podporované verze OpenFlow protokolu, nebo způsob komunikace pomocí API. Některé kontroléry umožňují správci nastavit velké množství technických detailů, jiné kontroléry mají technické detaily schované za graficky přehledným a srozumitelným rozhraním. Použitými kontroléry v této práci jsou Onos, OpenDaylight, Floodlight, Pox a Ryu.

1.3 Struktura práce

Práce je rozdělena do následujících kapitol:

- **Kapitola 2 - Rešerše problematiky:** v této kapitole se čtenář seznámí se základními principy moderních datových sítí. Jsou zde představeny typy síťových zařízeních a jejich funkcionalita a role v síti. Součástí této kapitoly je seznam známých komerčních a open-source zařízeních, popis používaných protokolů v moderních sítích a několik stran věnujících se popisu síťových akademií.
- **Kapitola 3 - Automatizovaný systém pro výuku:** Nosnou kapitolou celé práce je návrh a implementace samotného automatizovaného systému. Jsou zde definované požadavky na systém, způsoby komunikace jednotlivých komponent a diagramy pro popis funkcionalit systému. V této kapitole je umístěn i návrh vzhledu grafického rozhraní a popis použitých komponent při implementaci.
- **Kapitola 4 - Zadání pro praktická cvičení:** Aplikování systému na výuku je pomocí zadání pro praktická cvičení. Zadání obsahují konfigurační šablony systému. Součástí každé šablony je i sada testů, která hodnotí provedenou konfiguraci.
- **Kapitola 5 - Testování systému:** V této části práce jsou uvedeny metody testování softwaru k dosažení požadovaného cíle. Také se zde nachází provedené testy a jejich výsledky.
- **Kapitola 6 - Závěr:** V závěru jsou zhodnoceny dosažené výsledky práce a možnosti budoucího vývoje automatizovaného systému.

Kapitola 2

Rešerše problematiky

Kapitola *Rešerše problematiky* popisuje SDN technologii a prostředky pro provozování sítě v centralizovaném režimu správy a řízení sítě. V této kapitole je popsána i problematika emulace a virtualizace sítě a síťových funkcí.

2.1 Moderní datové sítě

Během desetiletí, kdy vznikal Internet tak, jak ho známe, vznikly potřeby pro nové aplikace a využití datových přenosů. S nově používanými aplikacemi se zásadně změnil charakter internetového provozu. Síťová infrastruktura se postupně vyvíjí od metalických k optickým rozvodům. V místě, kde není možné pokrýt oblast pevným síťovým připojením, nastupuje připojení bezdrátové. Nové aplikace a služby, stupeň a variabilita koncových zařízeních vyžadují flexibilní řešení [59].

S příchodem nových trendů na pole IT (Information Technology) se postupně začínají objevovat nové technologie v oblastech automatizace, digitalizace nebo orchestrace. Jednou ze základních myšlenek pro nově vznikající návrhy moderních datových sítí je otevřená standardizace, která může a nemusí být silovým a politickým procesem, a abstrakce síťových funkcionalit na jednotlivých úrovních správy sítě.

Dvěma základními koncepty pro moderní datové sítě jsou SDN (Software Defined Networking) a NFV (Network Functions Virtualization).

2.1.1 Softwarově definované sítě

SDN jsou stavebním blokem nového způsobu správy sítě typu IBN (Intent-Based Networking). Během nárůstu počtu uživatelů, zařízeních a distribuovaných aplikací v síti se současně i prostředí sítí stalo více komplexní. Nový způsob správy sítě přeměňuje hardwarově orientované sítě a manuální práci při konfiguraci síťových zařízeních na sítě řízené kontrolérem, které zachycují obchodní záměry a převádí je do zásad, které lze automatizovat a důsledně aplikovat v celé spravované topologii. Centrálním bodem aktivity v síti je kontrolér. Hlavní domény, ve kterých mezi sebou kontroléry spolupracují, jsou: přístupová část, WAN (Wide Area Network), data centrum a cloud [9].

IBN pracuje s těmito třemi základními funkčními bloky:

- Translation (překlad) - Zachycení záměru do zásad, díky kterým může síť reagovat na komunikaci.
- Activation (aktivace) - Instalace zásad do fyzické a virtuální síťové infrastruktury pomocí automatizace.
- Assurance (zajištění) - Využití analytiky a strojového učení k nepřetržitému sledování sítě k ověření dosaženého záměru nasazení.

2.1.2 Definice SDN

Stále častěji se objevují aplikace, které potřebují zasahovat do síťové infrastruktury a měnit její konfiguraci kvůli bezpečnosti, směrování síťových toků nebo *load-balancingu*. Zmíněné kroky jsou mnohdy nemožné z důvodu nutnosti konfigurace každého zařízení zvlášť. Na aktuální trendy reaguje nový přístup správy sítě, kterým je softwarově definovaná síť.

Nové trendy správy sítě vedou ke zvyšujícím se potřebám zejména na následující oblasti:

- přidávání a odebírání zařízení z topologie
- dynamická rezervace síťové kapacity
- rozvoj cloudových služeb
- zavádění nebo odebírání zásad napříč sítěmi
- nezávislost na výrobci síťového zařízení

Výše uvedené požadavky daly vzniknout novému konceptu *Software Defined Networking*. Tento koncept odděluje datovou vrstvu od řídicí vrstvy síťové infrastruktury. Síťový HW (hardware) neobsahuje řídicí SW (software), místo toho za něj logické rozhodování zpracovává kontrolér. K tomuto kontroléru se připojují všechna zařízení v dané síti. Tímto způsobem přecházíme od konceptu distribuovaného řízení k centralizovanému. Centralizovaný koncept se jeví výhodným z pohledu plánování a managementu, což ušetří především náklady na lidské zdroje pro správu sítě. Z důvodu redundance a zálohy je vhodné, aby v síti nebyla přítomna pouze jediná instance kontroléru, ale aby jich bylo více, kvůli zajištění vysoké dostupnosti řídicích funkcionalit.

Hlavním přínosem konceptu SDN je přímá konfigurovatelnost na základě oddělení datové a řídicí vrstvy, abstrakce zařízeních v síti, centralizace správy, otevřenost standardu bez závislosti na výrobci a přítomnost API (Application Programming Interface), což umožňuje uživateli komunikovat přes standardizované rozhraní s kontrolérem.

2.1.3 Dělení technologie podle principu a rozsahu použití

SD-branch

SD-Branch je nový způsob, jak rozšířit softwarově definované principy směrem k pobočkám. Vyznačuje se zjednodušeným hardwarem, vzdálenou centralizovanou správou a automatizací prostřednictvím programovatelnosti. Všechna místa, která integrují tento koncept, jsou spravovatelná vzdáleně z centrálního místa pomocí jediného řídicího systému. Další výhodou je možnost spuštění diagnostiky a následného provedení údržby napříč celým spravovaným úsekem sítě [14].

Z pohledu bezpečnosti jsou nasazovány pokročilejší technologie pro ochranu perimetru. Používají se NGFWs (Next-generation firewall), IPS (Intrusion Prevention Systems), nástroje na detekci malwaru nebo nástroje pro odražení DDoS útoku (Distributed Denial of Service). Nástrojem pro tento koncept je SD-WAN (Software-Defined Wide Area Network) [13].

SD-WAN

SD-WAN je softwarově definovaný přístup ke správě pobočkové sítě nebo WAN. Mezi hlavní výhody patří zjednodušení provozu sítě zásluhou automatizace a cloudové správy, optimalizace uživatelské zkušenosti a efektivity, zlepšení výkonu a agility aplikací a hlavně snížení nákladů v souvislosti s nezávislostí datových toků.

Tradiční funkcionalita WAN je zaměřená na propojování kampusů. K tomu je využito protokolu MPLS (Multiprotocol Label Switching). S příchodem cloudu je nutné řešit nové postupy, které budou reagovat na rychle se rozvíjející síť internetu. Postupně se rozšiřuje i použití širokopásmového a mobilního připojení.

Jedním z důvodů potřeby posílit infrastrukturu WAN je například přechod na SaaS (Software as a Service), IaaS (Infrastructure as a Service) a PaaS (Platform as a Service). Jedná se o nový přístup k používání služeb, infrastruktury a platformy napříč datovou síť, a to internetem. S otevřením internetu novým technologiím se současně vytváří i možné bezpečnostní hrozby a nároky na dodržování předpisů.

S příchodem SD-WAN je možné zjednodušit správu sítě díky centralizované WAN architektuře, která usnadňuje škálování koncových bodů, ať jsou zapojeny kdekoliv napříč spravovaným blokem sítě. Další výhodou je optimalizace výkonu uživatelských aplikací provozovaných v cloudu. V případě výpadku spojení je dynamicky upraven datový provoz mezi dedikovanými spoji pro zajištění bezvýpadkového provozu, který velmi často bývá označen jako kritický. Součástí této koncepce je i zaměření na bezpečnost, která je zajišťována na úrovni poboček. Komunikace s jednotlivými pobočkami je realizována pomocí šifrovaných tunelů typu VPN (Virtual Private Network) [7].

Platformy pro řízení datového toku existují v několika provedeních, a to od fyzických boxů, přes virtuální síťové služby, až po cloudové instance síťových zařízeních.

SD-Access

SD-Access (Software-Defined Access) je jedno z prvních síťových řešeních založených na záměrech podniků a institucí. Toto řešení poskytuje automatizované služby typu *end-to-end*, jako je segmentace, kvalita služby a analýza provozu pro uživatele, zařízení a pro

provoz aplikací. Současně automatizuje uživatelské zásady, tudíž organizace mohou zajistit odpovídající řízení přístupu na spravovaných koncových zařízeních. Práva a zásady jsou nastaveny pro uživatele, zařízení a konkrétní aplikace v celé síti. Nastavení funkcionality sítě v rámci organizace je řízeno pomocí centrálního prvku.

Výhodami SD-Access jsou automatizace pomocí *plug-and-play*, automatizovaná segmentace sítě a přiřazování zásad na skupiny a otevřená a programovatelná rozhraní řídicích prvků pro integraci s aplikacemi třetí strany [8].

2.2 Topologie SDN

V nově vzniklém konceptu datových sítí se objevují zařízení, která jsou označována jako *SDN compatible*. Tato zařízení mají rozdělenou infrastrukturu do logických vrstev - řídicí a datová. Veškerá pravidla provozu definuje kontrolér sítě. Pravidla jsou předávána z kontroléru na zařízení, které následně jedná již podle definovaných pravidel.

Centralizace řídicích funkcí má však i značné nevýhody. Vzniká zde SPOF (Single Point of Failure), což znamená, že v případě výpadku kontroléru nebo jeho síťové konektivity dojde k výpadku řízení celé sítě. Z tohoto důvodu je nutné provozovat řízení sítě v režimu vysoké dostupnosti HA (High Availability). V tomto režimu je provozováno několik kontrolérů současně, kde je zátěž řízení sítě rozdělena distribuovaně. Je nezbytné navrhnout redundantní konektivitu k těmto kontrolérům tak, aby v případě výpadku jedné linky nebyla zasažena celá síť.

Dalším důvodem pro umístění více kontrolérů v síti je snížení odezvy a zpoždění přenosu dat, zvýšení spolehlivosti a reakční schopnosti sítě. Tyto parametry sítě je nutné dodržet v určitých mezích, zejména pro takové aplikace, které podléhají SLA (Service Level Agreement). Komunikace mezi více kontroléry probíhá pomocí směrovacích protokolů OSPF, BGP a IS-IS.

V dnes spravovaných sítích se nejčastěji objevují již zmíněné kontroléry a přepínače, které jsou s SDN kompatibilní.

2.2.1 SDN kontroléry

V čele SDN stojí kontrolér. Znalost kontroléru pokrývá celou topologii sítě. Správa sítě kontrolérem spočívá v komunikaci s jednotlivými prvky sítě a v realizaci programování síťových funkcí. Datová síť s centrálním řídicím prvkem je označována jako centralizované řešení.

Kontrolér komunikuje se síťovými prvky pomocí otevřeného protokolu, kterým je nejčastěji OpenFlow 2.4. Pomocí komunikačního protokolu je na zařízeních sítě ovlivňováno směrování dat.

Z důvodu udržení vysoké dostupnosti řízení sítě je kontrolér provozován v clusteru nebo na virtuálním stroji. Těchto zařízení je provozováno v síti více z důvodu redundance.

Pro realizaci sítě podle SDN konceptu je nutné vybrat vhodný kontrolér kvůli definování správných metrik pro nově vytvořenou síť. Počet dostupných kontrolérů, které jsou k dispozici, neustále roste.

Komerční SDN kontroléry

Při výběru SDN kontroléru hraje významnou roli volba komerčního nebo open-source řešení. Řešení s otevřeným zdrojovým kódem sníží pravděpodobnost *vendor lockin*. Pro komunikaci se severním rozhraním se používají standardizované protokoly, jako je OpenFlow nebo OVSDB (Open vSwitch Database Management Protocol). Velká pozornost při výběru kontroléru by také měla být věnována funkčnosti, vhodnosti a podpoře samotného řešení, což nám ve velké míře zajišťují právě komerční kontroléry.

Dalším důležitým parametrem, který vychází vstříc komerčním řešením, je doba použitelnosti. Při výběru vhodného kontroléru je nezbytné porovnat počty běžících nasazení, oblasti pokrytí a případně informaci o tom, jak dlouho jsou tato nasazení funkční, Při porovnávání několika kontrolérů s podobnými vlastnosti může být velmi důležitý parametr právě doba provozu daného řešení.

Společností, které se dnes zabývají vývojem SDN kontrolérů pro komerční použití, je celá řada. Jsou to například Cisco, Brocade, Ericsson, Contrail, NEC, Sonus, Avaya, HPE, Huawei, Inocybe a další. Jejich seznam a podrobný přehled je dostupný v průzkumu společnosti SDXcentral ve spolupráci s jednotlivými výrobci [69].

Pro přiblížení některých významných společností na trhu a jejich kontrolérů byl použit výzkum od společnosti Miercom, která se zabývá zejména testováním technologií a konzultantskou činností [45].

Mezi tři porovnávané technologie ve zmíněném průzkumu patří SDN kontroléry společností Cisco, HPE-Aruba a Huawei. Tyto kontroléry jsou porovnávány v oblastech automatizace, segmentace, monitorování a správy sítě. Dalšími parametry, které jsou v porovnání diskutovány, jsou jednotná správa pro drátové a bezdrátové sítě, zjednodušené a automatizované operace, zabezpečení a virtualizace funkcí, způsob řešení problémů vzniklých v síti a intuitivní diagnostické nástroje.

Cisco Digital Network Architecture

V čele SDN stojí nástroj pojmenovaný DNA Center (DNAC), který obsahuje aplikace zaměřující se na design, zásady, nastavení, správu a ovládání síťových zařízeních.

Rozšiřující službou pro DNAC jsou bezpečnostní služby prvku ISE (Identity Services Engine), který zjednodušuje bezpečnostní správu zásad a poskytuje informace o uživatelích a zařízeních v síti. Služby ISE jsou abstraktně zobrazeny v DNAC. Administrátor má tedy základní přehled o dění a nastavení sítě včetně bezpečnostního přehledu na jednom místě.

SD-Access přístup poskytuje možnost virtualizování síťového provozu na fyzické infrastruktuře. Tento přístup je implementován v DNAC v podobně fabrikového řešení.

HPE-Aruba Mobile First Campus Solution

Společnost HPE byla rozšířena o bezdrátové technologie díky akvizici s firmou Aruba Networks. Bezdrátové technologie jsou zejména přístupové body, kontroléry a řídicí software. U této společnosti se nachází řešení pro SDN v podobě několika prvků, kterými jsou AirWave management, ClearPass a Mobility Master pro konfiguraci bezdrátové sítě a platformy Clarity, Network Analytics Engine, NetInsights a Cape Networks pro řízení a nastavování parametrů softwarově definované sítě.

HPE nabízí balíček správy sítě nazvaný Management Center, který je orientován na jádro sítě a datová centra, bohužel nikoliv na kampusové sítě, a je bez podpory bezdrátových

sítí. Pro nejlépe odpovídající porovnání s Cisco DNAC je vhodné porovnávat technologii AirWave, která je designována na mobilitu, proaktivní monitoring zařízení nebo uživatelů a aplikací pro udržení vysokého výkonu v síti.

Huawei Agile Campus Network Solution

Technologie pro sjednocené řízení podnikové sítě je pojmenována eSight. Je centrem pro kampusovou architekturu a současně je použita pro plánování a údržbu komplexní infrastruktury. Agile Controller zabezpečuje manipulaci se všemi síťovými operacemi napříč všemi vrstvami síťového modelu SDN, VLAN (Virtual Local Area Network) a se zásadami pro řízení komunikace.

Součástí agilního přístupu je SVF (Super Virtual Fabric). Tento nástroj je používán pro správu drátové a bezdrátové sítě a je provozován na přepínačích k tomu určených v nejvyšší vrstvě síťové topologie.

Z průzkumu celkově vychází s nejlepším hodnocením Cisco DNA Center. Hlavními důvody pro nejlepší hodnocení jsou schopnosti automatizace, rozšířená konfigurace, která vyžaduje čas jen v řádech desítek minut až hodin a vhodně implementovaný převod business požadavků a obsahu na automatizované síťové nastavení. DNAC nabízí jednoduchou správu zásad a správu klientů připojených jak drátovou, tak bezdrátovou technologií. Výhodou je i přítomnost *dashboardu*, který je velmi propracovaný a poskytuje rychlý a jednoduchý přehled o chování celé infrastruktury.

Open-source SDN kontroléry

Open-source kontrolérů existuje celá řada. Mezi nepřeberným množstvím, které trh v současné době nabízí, je velmi obtížné určit vhodný řídicí prvek pro datovou síť. Ke správnému rozhodnutí výběru SDN kontroléru zpracovávají různé technologické organizace srovnání a hodnocení jejich výkonů [40].

Pro výběr vhodného open-source kontroléru je nutné zpracovat průzkum. Během průzkumu je nezbytné najít veškeré dostupné informace o SDN kontrolérech napříč literaturou, webovými stránkami, blogy a dalšími dostupnými zdroji. Po průzkumu, který byl proveden IT odborníky z Fraunhoferova institutu pro bezpečné informační technologie, vyplynulo, že největšími hráči mezi kontroléry na poli SDN jsou: POX, Ryu, Trema, FloodLight a OpenDaylight [63].

POX je kontrolér, který je odvozený od kontroléru NOX. Jedná se o prvek sítě, který je napsaný v programovacím jazyce Python. Je používán pro průzkum a ladění SDN sítě, virtualizaci síťových funkcí, návrh designu kontroléru a k definování programovacích modelů.

Ryu je podporován NTT (Nippon Telegraph and Telephone Public Corp.). Je založený na systému komponent komunikujících vzájemně mezi sebou. Velké množství jich je předdefinovaných v základní instalaci. Tyto komponenty mohou být upravovány, rozšiřovány a skládány do větších celků pro přizpůsobení kontroléru dané aplikaci. K vývoji komponent je možné použít libovolný programovací jazyk. Základním jazykem, ve kterém je napsáno jádro kontroléru, je Python.

Trema je podporován NEC labs (Nippon Electric Corporation) a jeho nejdůležitější vlastností je snadno zapisovatelný kód a vysoký výkon. Skriptovací jazyk je Ruby, který je použit pro zvýšení produktivity při práci s kontrolérem. Pro zvýšení výkonu se používá

kompilátor psaný v jazyce C. Trema je současně *framework*, díky kterému je možné sestavit vlastní SDN kontrolér v jazyce Ruby nebo C. Součástí tohoto *frameworku* jsou základní knihovny pro vytvoření OpenFlow kompatibilních přepínačů.

Floodlight je SDN kontrolér, který vznikl z původního kontroléru Beacon vyvinutého na Stanfordské univerzitě. Skládá se z celé řady modulů, kde každý z nich poskytuje služby ostatním modulům. Řídicí logiku moduly poskytují pomocí rozhraní Java API a REST API (Application Programming Interface). Kontrolér je podporován architekturami operačních systémů Linux, Windows a Mac.

OpenDaylight je kontrolér, který si klade za cíl pokrýt většinu hlavních komponent architektury SDN pomocí robustního kódu. Okolo tohoto kontroléru je vybudována rychle rostoucí komunita, která přispívá k vývoji kódu. Jednotlivá odvětví kódu jsou používána pro komerční produkty.

Technologie všech SDN kontrolérů, které jsou požity pro **severní rozhraní**, jsou REST/JSON (Representional State Transfer/JavaScript Object Notation), Java/RPC (Remote Procedure Calling), OSGi (Open Services Gateway initiative) a Quantum.

Technologie pro **jižní rozhraní** mohou být rozděleny do dvou základních skupin, kterými jsou řízení a kontrola.

Řídicí technologie jsou OVSDB , OF-Config (OpenFlow-Config), SNMP (Simple Network Management Protocol) a XMPP (Extensible Messaging and Presence Protocol).

Mezi kontrolní technologie patří OpenFlow, XMPP, PCE/PCEP (Path Computation Element/Path Computation Element Communication Protocol), FoRCES (Forwarding and Control Element Separation), I2RS (Interface to Routing System), BGP (Border Gateway Protocol) a BGP-LS (Border Gateway Protocol Link-State) [63].

Výběr kontroléru pomocí jednoho kritéria je triviální. Výběr pomocí několika kritérií se již jedná o multikriterionální rozhodovací problém (Multi-Criteria Decision Making). Pro řešení takového problému existuje několik metod, v tomto případě bylo použito řešení pomocí metody Analytický Hierarchický Proces (Analytic Hierarchical Process).

Metoda AHP používá párovou prioritizaci a má integrovaný mechanismus pro kontrolu konzistence. Pro správné použití této metody je nutné namapovat hodnoty vlastností na párovou prioritizační škálu. Adaptace je vytvořena pomocí monotónických interpolačních a extrapolačních mechanismů.

Základními parametry, podle kterých lze kontroléry srovnávat, jsou:

- programovací jazyk, ve kterém je kontrolér napsaný
- podporovaná síťová architektura
- způsob komunikace severního rozhraní
- podporovaný komunikační protokol jižního rozhraní
- podpora TLS (Transport Layer Security)
- způsob licencování
- podporovaná platforma

- existence dokumentace a její kvalita

Z použití metody MCDM, a to konkrétně AHP aplikované pomocí mapovacího mechanismu, vychází podle odborníků z Fraunhoferova institutu ve výše zmíněné pětici vybraných SDN kontrolérů nejlépe kontrolér Ryu [63].

2.2.2 SDN přepínače

Přepínač je síťový prvek, který propojuje k němu připojená zařízení do hvězdicové topologie. Tradiční přepínač pracuje nezávisle na zbytku sítě. Oproti tomu SDN přepínač má logicky oddělenou datovou a řídicí rovinu. Pokud SDN přepínač poprvé přijme data, pro která nemá uložené instrukce k jejich zpracování, kontaktuje SDN kontrolér. Zpět dostane informaci o tom, jak má daný tok dat zpracovat. Tuto informaci si přepínač uloží do své paměti a následně při přijetí dat se stejnou hlavičkou není potřeba kontaktovat kontrolér. SDN přepínač a SDN kontrolér používají nejčastěji pro komunikaci protokol OpenFlow. Informace v této kapitole jsou čerpány z těchto pramenů [80] a [18], pokud tomu není uvedeno jinak.

Softwarové SDN přepínače

Softwarový přepínač je virtuální přepínač implementovaný v podobě softwaru na zařízení. Je používán pro zjednodušení komunikace mezi zařízeními. Tento typ přepínače může být navrhnut pro komunikaci mezi virtuálními prostředími nebo jako firmware do hardwarového přepínače. V tabulce 2.1 je uveden přehled nejznámějších softwarových SDN přepínačů.

Název přepínače	**Společnost**	**Licence**
B4N SwitchOS	Brain4Net	komerční
OcNOS	IP Invision	komerční
PF1000	NEC	komerční
VortiQa Switch	NXP Semiconductors	komerční
XorPlus/PicOS	Pica8	komerční
ZeroTier Switch	ZeroTier	komerční
BESS	Berkeley NetSys Lab	open-source
bmv2	P4	open-source
Indigo	Project Floodlight	open-source
Lagopus	Lagopus	open-source
LINC	FlowForwarding	open-source
ofsoftswitch13	Ericsson, CPqD	open-source
Open vSwitch	Open Community	open-source
OpenContrail vSwitch	Tungsten Fabric	open-source
OpenFlow Reference	Stanford	open-source
OpenFlowClick	Yogesh Mundada	open-source
Pantou/OpenWRT	Stanford	open-source
Snabb Switch	Snabb	open-source
Switch Light	Big Switch Networks	open-source
VPP	FD.io	open-source

Tabulka 2.1: Přehled softwarových SDN přepínačů

Hardwarové SDN přepínače

Přepínání dat pomocí hardwarově designovaného zařízení je prováděno funkcí jednotlivých čipů, které jsou připájené k základní desce síťového zařízení, případně k ní připojeny pomocí slotu datové sběrnice. Některé komponenty je možné v přepínači *upgradovat*, jiné jsou fixní. Hardwarový přepínač disponuje několika typy pamětí, které jsou podle svého určení různého fyzického charakteru. Přepínače s podporou SDN extrahují svou řídicí funkcionalitu na SDN kontrolér. V tabulce 2.2 je uveden přehled nejznámějších hardwarových SDN přepínačů.

Společnost	**SDN kompatibilní produkty**
Alcatel-Lucent	OmniSwitch – 9900, 6900, 6865, 6860, 6560, 6465, 6450, 6350
Allied Telesis	AT-x230, AT-x510, AT-x310, AT-x930 families, ATDC2552XS
Arista	7280SE, 7050, 7050X, 7500 families
Aruba	5400R, 3810, 2930M, 2930F, 2920, 2540, 2530
Brocade	VDX 2741, VDX 6740, VDX 6940, VDX 8770, NetIron CES 2000, NetIron CER 2000, ICX 6430, ICX 6450, FCX, ICX 6610, ICX 6650, FSX 800/FSX 1600, ICX 7250, ICX 7450, ICX 7750
Centec	V580, V350, V330, V150
Cisco Systems	C9600, C9500, C9400, C9300, C9200, C6800, C6500, C4500E, C3850, C3650, C3560-CX, C2960-L, Nexus 9000 series, Nexus 3000 Series, Nexus 31128PQ, Nexus 3232C, Nexus 3264Q
Dell	N4000, N3000, N2000, N1500, N1100
Edgecore Networks	AS7712-32X, AS6812-32X, AS5812-54T, AS5712-54X, AS5710-54X, AS5600-52X, ECS4620, ECS4510, ECS4210, ECS4120, ECS4100, ECS3500
Extreme Networks	Summit X440, X460, X460-G2, X480, X670, X670-G2, X770
H3C	S6800-4C, S6800-4C, S6800-2C, H3C S6800-32Q, H3CS6800-54QT, H3CS5130-54S-HI, H3CS5130-54CPWR-HI, H3CS5130-54C-HI, H3CS5130-34C-HI
HPE	2920, 3500, 3800, 5400, 6200, 6600 , 8200, FlexNetwork 5130, FlexNetwork 7500, FlexFabric 5700 series, Flex- Fabric 7900
Huawei Juniper Networks	S5720, S6720-EI, S7700, S9700, S12700 series switches EX9200, MX80, MX240, MX480, MX960, MX2010, MX2020, QFX5100
NEC	PF5248, PF5240
Pica8	P5401, P5101, P3930, P3922, P3297
Quanta	BMS T3048-LY9, T3048-LY8, T3048-LY2R, T3048-LY2, T5032-LY6, T1048-LB9, T3040-LY3, T5016-LB8D

Tabulka 2.2: Přehled hardwarových SDN přepínačů [68]

2.3 Srovnání open-source a komerčního řešení

Při volbě způsobu řízení sítě, zda SDN kontrolérem nebo zatím standardním klasickým způsobem, je nutné si definovat požadavky, které jsou pro každou organizaci klíčové. Tyto požadavky mohou hrát roli nejen v otázce výkonu a funkcionality daného řešení, ale také v otázce lidských zdrojů. Pokud se v organizaci nachází správce se znalostmi open-source kontrolérů, je třeba dbát i na přítomnost dalšího správně s dostatečnou znalostí kvůli zajištění zastupitelnosti. V případě komerčního řešení při nasazení SDN je k dispozici většinou 24/7 podpora pro řešení neobvyklých problémů a případné výměně zařízení.

Open-source i komerční řešení jdou značným tempem kupředu, avšak výběr se musí důkladně promyslet, protože se s největší pravděpodobností bude jednat o způsob řízení sítě na několik let dopředu po jeho nasazení.

2.4 Protokoly používané v SDN

Komunikace mezi datovou a řídicí vrstvou je většinou proprietární záležitost a je definována výrobcem. Velice často není zajištěna kompatibilita mezi zařízeními různých výrobců, například jde o CLI (Command Line Interface). Definování standardizovaných protokolů pro komunikaci mezi datovou a řídicí rovinou je zásadní pro SDN. Definování takovýchto protokolů pracujících s API jednotlivých vrstev umožňuje interoperabilitu mezi zařízeními odlišných výrobců.

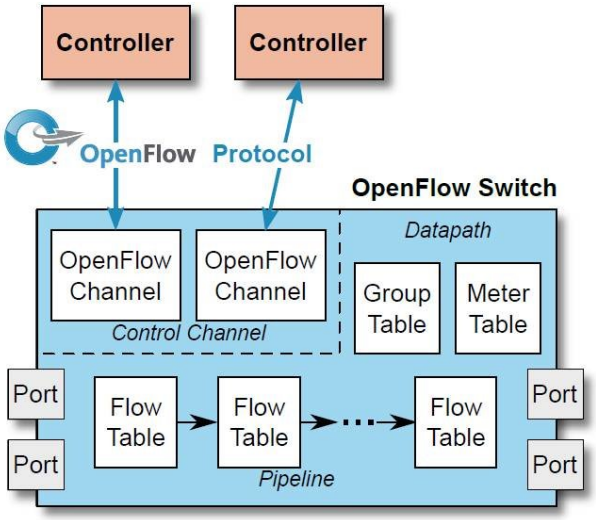
Protokoly je možné rozdělit do dvou základních kategorií podle způsobu jejich funkčnosti. Protokoly OVSDB (Open vSwitch Database Management Protocol) a OF-Config (OpenFlow Config) jsou používány pro zapnutí a vypnutí portu, případně pro vytvoření logického tunelu. Oproti tomu protokol OpenFlow slouží k programování toku paketů v síťovém zařízení [71].

OpenFlow

OpenFlow je protokol 2. vrstvy ISO/OSI (International Organization for Standardization/Open Systems Interconnection model) modelu. OpenFlow zprostředkovává komunikační propojení mezi *forwarding plane* síťového zařízení a kontrolérem. Jedná se o nejrozšířenější otevřený standard jižního rozhraní. Komunikace probíhá převážně pomocí zabezpečeného protokolu založeného na TLS při využití asymetrické kryptografie [56]. Zařízení, která jsou kompatibilní s OpenFlow protokolem, se dělí na dvě skupiny:

- *OpenFlow-only* - Zařízení s tímto označením podporují zpracování paketů pouze pomocí *OpenFlow pipelines* a nedovolují žádný jiný způsob zpracování dat.
- *OpenFlow-hybrid* - v hybridních zařízeních je dovoleno zpracovávat data jak pomocí OpenFlow operací, tak pomocí standardních L2 přepínacích operací, která jsou tradičně nakonfigurována na daném zařízení [19].

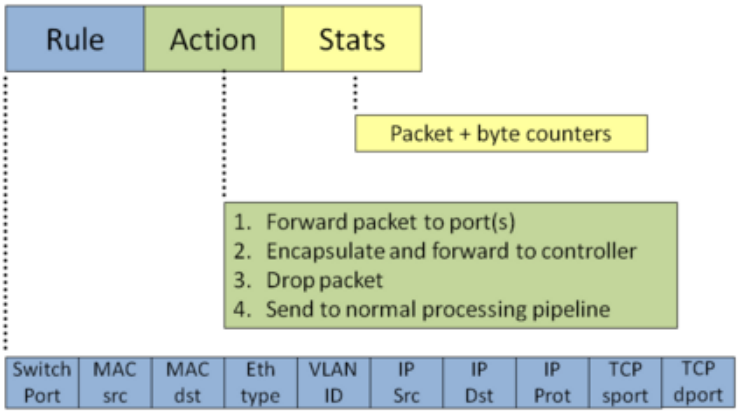
Na obrázku 2.1 jsou schematicky znázorněny komponenty OpenFlow přepínače. Je zde zviditelněn průběh toku paketů skrze OpenFlow tabulky a pravidla, která jsou v nich umístěna.



Obrázek 2.1: Komunikace kontroléru a přepínače pomocí protokolu OpenFlow [66]

OpenFlow protokol dovoluje SDN kontroléru řídit komunikaci probíhající na OpenFlow přepínači. Základní řídicí instrukce jsou definování, úprava a mazání datových toků. Datovým tokem je označena sada pravidel, která se váže ke všem paketům, které do tohoto toku patří. Přepínač zpracuje pakety z daného toku a rozhodne, zda se předají na jeden nebo více portů, zahodí se nebo se předají kontroléru pro další zpracování výjimek [29].

Každý datový tok obsahuje pole shody paketů, prioritu paketů, různé čítače, pokyny pro zpracování paketů, časové limity a další. Tato pravidla jsou uchovávána v tabulkách (viz obrázek 2.2). Každý paket, než opustí přepínač výstupním rozhraním, může být zpracován pomocí souboru pravidel napříč OpenFlow tabulkami [66].



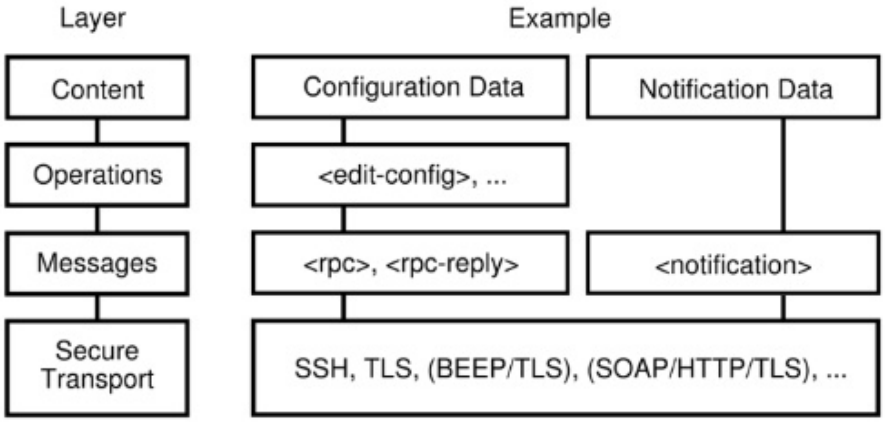
Obrázek 2.2: Definice záznamu v OpenFlow tabulce [17]

OpenFlow protokol byl navržen jako *multivendor*, což znamená, že je možné ho provozovat na zařízeních různých výrobců. Standard OpenFlow protokolu je definován organizací ONF (Open Networking Foundation) pro implementaci SDN v síťových zařízeních.

OpenFlow protokol byl vydán v první verzi 1.0 v roce 2009. Postupně byly přidávány různá vylepšení a úpravy až do roku 2015, kdy byla vydána doposud poslední verze 1.5.

NETCONF

NETCONF (Network Configuration Protocol) je protokol definovaný v RFC 6241 [20]. Jedná se o protokol založený na XML (Extensible Markup Language) pro RPC (Remote Procedure Call). Tento mechanismus se používá pro vzdálenou správu zařízeních, jako je instalace, manipulace a mazání konfigurací. Podporuje komunikaci založenou na transakcích a síťovou komunikaci napříč několika zařízeními.



Obrázek 2.3: Vrstvy protokolu NETCONF [31]

Protokol NETCONF definuje architekturu následujícími vrstvami:

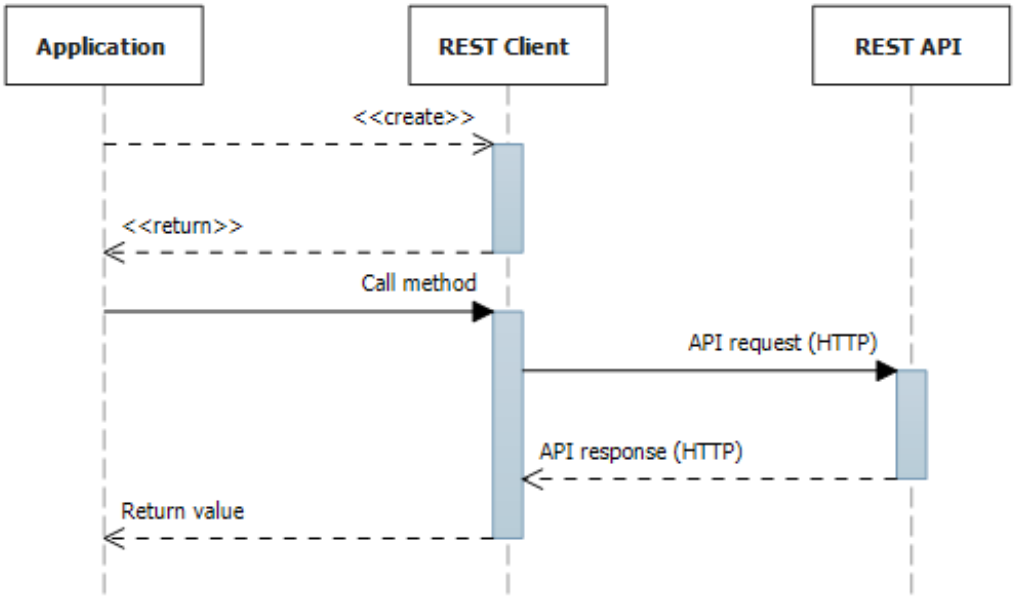
- *Content layer:* Dovoluje konfiguraci zařízení a uložení dat notifikací.
- *Operations layer:* Definuje sadu operací pro manipulaci s daty v datovém úložišti.
- *RPC Messages layer:* Podpora vzdáleného volání procedur a notifikací.
- *Secure Transport layer:* Vrstva je založená na TCP (Transmission Control Protocol) komunikaci pro spolehlivý přenos informací. Pro šifrování je využito SSH (Secure Shell) nebo TLS.

REST

REST (Representational State Transfer) API se stalo standardem pro komunikaci s webovými službami, protože je flexibilní, škálovatelné a multiplatformní. REST je založen na modelu HTTP/S (HyperText Transfer Protocol/Secure) dotaz-odpověď v komunikaci klient-server.

REST používá následující metody pro komunikaci klient-server:

- GET - Získání dat z cesty, která je specifikovaná v URL (Uniform Resource Locator).
- POST - Zápis dat na místo určené cestou v URL. Metoda se používá pouze pro zápis nových dat.
- PUT - Nahrazení dat definovaných existující cestou v URL. Nemůže být použito pro vytvoření doposud neexistujících dat.
- DELETE - Smazání existujících dat z cesty definované v URL.



Obrázek 2.4: Komunikace aplikace se serverem pomocí REST API [30]

Komunikace s API rozhraním probíhá nejčastěji pomocí JSON objektů. Jedná se o otevřený standard formátu souborů pro výměnu dat, který používá lidsky čitelnou podobu. Datové objekty jsou dvojice `atribut:hodnota`. Jedná se o běžně používaný formát v různých aplikacích [15]. Dnes nahrazuje formát XML. Na obrázku 2.4 je znázorněna komunikace mezi aplikací a serverem pomocí REST API.

SNMP

SNMP (Simple Network Management Protocol) se využívá pro práci s *management* vrstvou na rozdíl od OpenFlow, který pracuje s řídicí vrstvou. SNMP protokol je definován v RFC 1157 [5].

Tento protokol je možné použít k monitorování stavu síťových zařízeních. V omezené míře je možné využití k jednoduchému nastavení funkcionality zařízení. Sbíraná data o zařízeních jsou organizována v MIB (Management Information Base) struktuře. Protokol prošel od

svého zavedení několika změnami od verze 1, přes verzi 2 až k dnešní verzi 3. Existuje další řada vylepšení a rozšíření funkcionalit definovaných v navazujících standardech RFC.

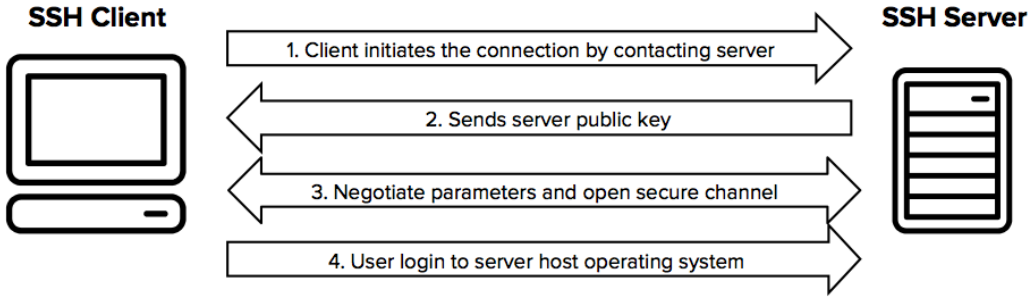
Protokol používá na straně spravovaného zařízení agenta, což je software, pomocí kterého jsou získávány informace ze síťového zařízení. Informace jsou uloženy na zařízení NMS (Network Management Station), kde je s nimi dále pracováno.

V případě vyhodnocení stavového hlášení přesahující uživatelsky nastavenou mez je upozorněn správce na anomálii v síti.

SSH

SSH je protokol, který se používá pro vzdálené šifrované spojení mezi síťovými zařízeními. Poskytuje několik úrovní zabezpečení komunikace šifrováním přenášených dat. SSH je nástupcem nezabezpečených protokolů jako je například telnet, rlogin nebo FTP. Nejčastější použití tohoto protokolu je pro poskytnutí zabezpečeného přístupu uživatelů nebo pro automatické procesy, interaktivní a automatizovaný přenos dat, zadávání vzdálených příkazů a pro správu síťové infrastruktury.

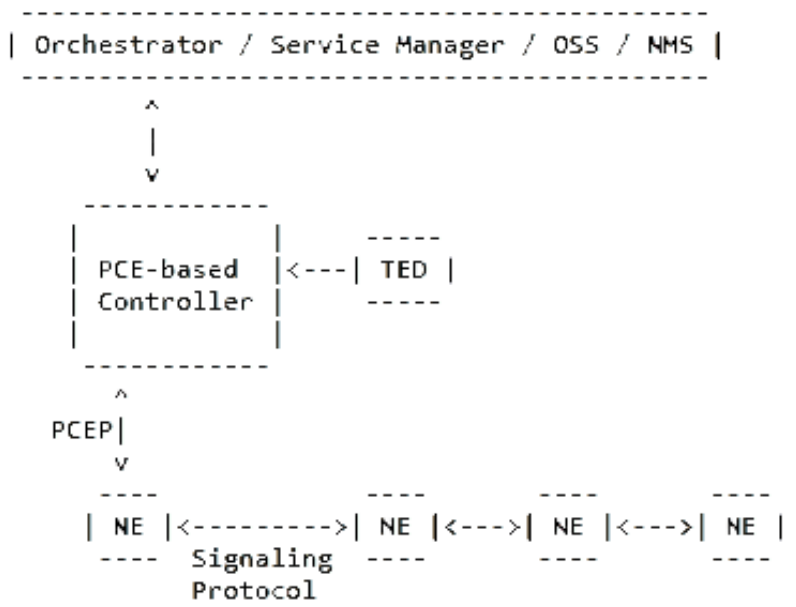
Protokol pracuje v modelu klient-server, což znamená, že komunikace je započata na straně klienta. Klient zahájí komunikaci odesláním požadavku na kontaktování serveru. Následně je pomocí veřejného klíče serveru ověřena jeho identita. Po počáteční fázi nastavení je použit symetrický šifrovací klíč a hashovací algoritmus pro zajištění bezpečnosti a integrity přenášených dat mezi klientem a serverem [74].



Obrázek 2.5: Funkcionalita SSH protokolu [74]

PCEP

PCEP je definovaný v RFC 5440 [78]. Jedná se o jeden z nejrozšířenějších řídicích protokolů pro komunikaci mezi centrálním kontrolérem a síťovými zařízeními. Pracuje se severním rozhraním zařízení. Umožňuje práci PCE, hlavnímu řídicímu kontroléru, v síti zmapovat všechna zařízení, která jsou v dané SDN doméně spravována. V dalším rozšiřování funkcionality PCE je umožněno určení cest mezi kontrolérem a zařízeními pomocí statických LSPs (Labeled Switched Paths), segmentového směrování nebo pomocí spojování servisních funkcí. Na obrázku 2.6 je zobrazena topologie, ve které figuruje tento řídicí protokol.



Obrázek 2.6: Architektura centrálního kontroléru s řídicí vrstvou [1]

Centrální kontrolér je požádán síťovou komponentou OSS (Operational Support System) o vytvoření konektivity. PCE kontrolér spočítá cesty v síti pomocí síťové topologie, dostupných zdrojů a dalších informací podporovaných sítí. Informace jsou uchovávány v TED (Traffic Engineering Database). Následně PCE odešle požadavky na NEs (Network Elements) pomocí protokolu PCEP. V dalším kroku je použita řídicí vrstva, pomocí které se naváže požadované spojení a rezervují se síťové prostředky [1].

2.5 Virtualizace sítě

Virtualizace sítě je v oblasti výpočetní techniky kombinování hardwarových a softwarových síťových zdrojů a síťových funkcí do jedné softwarové administrativní entity, virtuální sítě. Virtualizace zahrnuje virtualizaci platformy doplněnou virtualizací zdrojů. Virtualizovat je možné plochu vzdáleného počítače, síť pro rozdělení šířky pásma mezi nezávislé kanály, software pro oddělení aplikací od hardwaru a operačního softwaru nebo úložiště pro kombinování několika prostředků v jediné úložné zařízení [42].

Virtualizace sítě poskytuje běh několika virtuálních propojeních na společné síťové infrastruktuře. Každá virtuální síť je soubor virtuálních uzlů a virtuálních propojů označovaných jako *Network Virtualization Environment* (NVE). Jednou z nejdůležitějších vlastností virtuální sítě je podpora současně běžících protokolů a služeb pro konkrétní aplikaci [21].

Technologie virtualizace datové sítě odděluje role tradičních poskytovatelů internetových služeb ISP (Internet Service Provider) na dva poskytovatele. Poskytovatele infrastruktury InPs (Infrastructure Providers), kteří spravují fyzickou topologii, a na poskytovatele služeb

SPs (Service Providers), kteří vytváří virtuální sítě pomocí agregace zdrojů a následně nabízí koncovým zákazníkům síťové služby [6].

Virtualizace a emulace

Virtualizace vytváří vrstvu mezi hardwarem a softwarem. Virtualizační platforma následně řídí přístup k fyzickým zdrojům hostitele, což umožňuje předávání těchto zdrojů hostujícím strojům. Každý hostující stroj je postaven na abstraktní vrstvě označované jako hypervisor. Hypervisor upravuje množství a způsob komunikace hosta s fyzickými zdroji.

Emulace umožňuje vytvoření prostředí pro operační systém. Na jednom systému je současně možné emulovat hardwarové zdroje pro systém jiný. Takto vytvořené prostředí se chová jako *„hardware-like“*, avšak systém nemá přímý přístup k samotným HW prostředkům stroje, na kterém je spuštěn podkladový systém pro tuto emulaci [39].

Prvky sítě

Prvky virtuální sítě mohou do určité míry plně nahrazovat funkcionalitu HW komponent běžících odděleně, avšak s většími či menšími problémy se správnou konfigurací jednotlivých funkcionalit. Jednotlivé prvky sítě můžeme pojmenovat následně:

- **virtuální stroj** - VM (Virtual Machine) nebo-li host, je koncové zařízení, které obsahuje lokální jmenný prostor definovaný pro konkrétní operační systém, případně pro sadu funkcí a služeb, které má virtuální stroj v síti zastávat.
- **virtuální přepínač** - Virtuální přepínač stejně jako fyzický přepínač dovoluje spolu propojit jednotlivé síťové komponenty. Hlavními výhodami virtuálního přepínače jsou jednoduchá migrace a nasazení virtuálních serverů, možnost správy virtuálních přepínačů nasazených pomocí hypervisoru, a ve srovnání s fyzickým přepínačem lze snadno zavést nové funkce, které mohou souviset s hardwarem nebo firmwarem.
- **síťový most** - Síťový most (bridge) dovoluje připojit virtuální stroj do LAN (Local Area Network) pomocí hostitelského počítače. Propojuje virtuální síťový adaptér virtuálního stroje s fyzickým ethernetovým adaptérem hostitelského počítače. Bridge může být současně chápán jako síťové zařízení, které je historicky považováno za předchůdce přepínače ve smyslu logického přepínaní rámců.
- **NAT** - NAT (Network Address Translation) dovoluje připojit virtuální zařízení do internetu pomocí jedné IP (Internet Protocol) adresy, která je používána hostovaným zařízením.
- **DHCP** - DHCP (Dynamic Host Configuration Protocol) server je služba, která poskytuje IP adresy a nastavení pro virtuální zařízení, která nejsou přímo připojena do internetu.

2.6 Vzdělávací síťové akademie

Motivace

Motivací pro společnosti zaměřující se na IT technologie je k zakládání a udržování akademií pro systémové inženýry, zvýšení populárnosti konkrétních technologií a zvýšení technické odbornosti zaměstnanců ve svých technických týmech.

IT společnosti musí také reagovat na znalostní povědomí nově vystudovaných inženýrů z vysokých škol, které zatím všechny nemají dostatečně upraveny tematické plány v reakci na současné trendy. Pomocí vzdělávacích akademií si tito noví zaměstnanci doplňují znalosti, které jim schází. Akademie neslouží pouze pro vzdělávání nových pracovníků, ale také pro obnovení aktuálních informací senior systémových inženýrů.

Možnosti akademií

Vzdělávací akademie nabízejí jak svým zaměstnancům, tak veřejnosti, výukové kurzy a certifikace. Výukové kurzy jsou nejčastěji dostupné online v podobně *e-learningu*. Pro některé kurzy je nutná přítomnost lektora a místnost, ve které jsou k dispozici HW zařízení, na kterých probíhá výuka. Některé kurzy zahrnují po jejich dokončení možnosti certifikace, což je závěrečná zkouška, která je většinou mezinárodně uznávaná a dokládá znalosti studenta o dané problematice.

Ne všechny společnosti mají k dispozici prostory, čas a zdroje na provozování akademií, a proto tyto společnosti využívají sdílené učebny nabízené pro setkávání techniků a vzdělávání. Mimo firem, které mají svá síťová řešení, jsou zde i firmy, které se zaměřují pouze na vzdělávání a školení. Školení mohou být jak v síťové laboratoři, tak i online formou.

Akademie jsou velmi často provozovány v neziskových organizacích, jako jsou například školy a jiné vzdělávací instituce. Technologické firmy nabízí těmto institucím statut vzdělávací akademie za předpokladu, že splní určité podmínky. Mezi podmínky nejčastěji patří dostatečně velké a dostupné prostory, přítomný lektor, který má akademii na starosti, a mnohdy je ve smlouvě o spolupráci uveden i minimální počet proškolených a otestovaných osob za určité časové období. Lektoři působící v těchto akademiích musí mít velmi často složenou zkoušku, která jim dovoluje vyučovat a certifikovat vyučovaný kurz.

Aktuální problematika

Mezi aktuálně nejpopulárnější kurzy patří IT bezpečnost a cloudové technologie. Popularitu kurzů určuje, jak poptávka po kurzech, tak aktuálnost problematiky, na kterou musí technologické firmy aktivně reagovat.

Bezpečnost nebyla vždy od zrodu internetu na prvním místě, ale posledních pár desítek let je nezbytně nutné se jí věnovat čím dál více. Z pohledu cloudu je poskytováno více služeb, které jsou nabízeny v podobě SaaS, IaaS a PaaS. Z pohledu SDN je stále rostoucí síť potřeba efektivně spravovat a monitorovat. Aktuální kurzy, které jsou dostupné napříč vzdělávacími akademiemi, bohužel zatím neposkytují adekvátní kurzy, které by na tuto problematiku dostatečně reagovaly. Tato diplomová práce se zabývá tím, jak pomoci vzdělávacím akademiím s výukou SDN, více v kapitole 3.

Společnosti poskytující vzdělávací akademie

Společností, které poskytují ve svém portfoliu vzdělávací akademie, je celá řada. Pouze však malá část z nich má své kurzy zaměřené na síťové technologie. V tabulce 2.3 je uveden přehled společností, které mají část ze svých kurzů zaměřené na síťovou problematiku. Společnosti jsou v tabulce seřazeny podle množství poskytovaných kurzů.

Společnost	**Odkaz na akademii**
Cisco Systems	https://www.netacad.com/
Juniper Networks	https://learningportal.juniper.net/
Huawei Technologies	https://e.huawei.com/cn/talent
Extreme Networks	https://www.extremenetworks.com/education
Aruba Networks	https://academy-aruba.talentlms.com/
HPE	https://certification-learning.hpe.com/
Avaya	https://www.avaya-learning.com/
MikroTik	https://mikrotik.com/training/
EC-council	https://www.eccouncil.org/academia/
Intel Corporation	https://builders.intel.com/
Microsoft	https://www.microsoft.com/learning/ https://www.microsoftacademy.cz/
IBM	https://www.ibm.com/services/learning/
Google	https://skillshop.withgoogle.com/
Check Point Software Technologies	https://training-certifications.checkpoint.com/
F5 Networks	https://f5.com/education
Palo Alto Networks	https://www.paloaltonetworks.com/.. ../services/education/academy
GNS3 Academy	https://gns3.teachable.com/

Tabulka 2.3: Přehled společností poskytující vzdělávací akademie

Kapitola 3

Automatizovaný systém pro výuku

Systém pro automatizaci výuky bude sloužit zejména středoškolským a vysokoškolským studentům síťových akademií, kteří budou využívat připravené šablony pro spuštění virtuální topologie obsahující síťová a koncová zařízení, virtuální propoje a SDN kontroléry. Tento systém bude průběžně hodnotit práci studentů během plnění dílčích úkolů daného zadání.

V této kapitole se diplomová práce současně zaměřuje na porovnání dostupných technologií vhodných pro implementaci automatizovaného systému pro výuku SDN technologií.

3.1 Cíle řešené problematiky

Pro návrh systému je nutné se zaměřit na porovnání a zhodnocení dostupných technologií a určit rozsah práce tak, aby vyhovoval potřebám výuky.

Systém by měl být vzdělávacího charakteru a poskytovat studentům zpětnou vazbu na provedenou konfiguraci síťových zařízeních definovaných v zadání. Student nakonfiguruje požadovanou funkcionalitu definovanou v zadání a systém automatizovaně ohodnotí správnost řešení.

Dalším cílem, který by měl systém splňovat, je usnadnění přípravy výuky. Ze strany učitele je nutné pro konkrétní tematický okruh předmětu připravit hodinu tak, aby byla pro studenty co nejpřínosnější, a současně aby výuka proběhla ideálně bez náhodně vznikajících problémů. Pro zjednodušení práce učitele, by měl systém podle připravené šablony spustit virtuální síť a vybraný SDN kontrolér. Tato šablona by měla být editovatelná podle potřeb výuky.

3.2 Přínosy pro uživatele

- Jednoduše spustit topologii podle předem připravené šablony.
- Spustit vybraný SDN kontrolér.
- Provést požadovanou konfiguraci síťových zařízeních.
- Získat zpětnou vazbu na provedenou konfiguraci topologie.

3.3 SWOT analýza

SWOT analýza poskytuje rozbor a hodnocení vnitřního a vnějšího prostředí. Do vnitřního prostředí řadíme silné stránky (Strengths) a slabé stránky (Weaknesses). Do vnějšího prostředí řadíme příležitosti (Opportunities) a hrozby (Threats).

Silné stránky

- Automatizované vytvoření virtuální topologie
- Možnost přípravy šablony pro topologii před začátkem hodiny nebo školení
- Práce s různými SDN kontroléry
- Přehledné uživatelské rozhraní
- Oddělené běhové prostředí od podkladového systému pro potřeby síťové laboratoře
- Absence finančních nákladů

Slabé stránky

- Celková náročnost realizace
- Při velké topologii nadměrný nárůst využitých HW zdrojů
- Potřeba reagovat na změny v API rozhraních SDN kontrolérů a dalších použitých nástrojů

Příležitosti

- Snadné rozšíření modulárního systému o další funkcionality plynoucí z testování systému
- Možnost rozšíření použití systému i pro HW SDN přepínače a kontroléry
- Zajištění kontinuální výuky pro problematiku SDN díky šablonám topologie a virtualizaci
- Snadné zavedení výuky SDN technologií

Hrozby

- Slabá komunita kolem některých open-source SDN kontrolérů a nástrojů pro virtuální topologii
- Existence konkurenčního řešení
- Nedostatečné HW prostředky na starších strojích nepodporující virtualizaci

3.4 Koncepce systému

Automatizovaný systém je distribuován v podobě virtuálního stroje spolu s dalšími komponentami, které využívá. Tento virtuální stroj je možné spustit na libovolné platformě. Hlavní výhodou použití virtuálního stroje je možnost tvorby *snapshotů* v klíčových stavech při práci s aplikacemi ve virtuálním prostředí.

Ve virtuálním stroji je přítomna aplikace automatizovaného systému pro práci s virtualizovanou sítí a SDN kontrolérem. Tento automatizovaný systém poskytuje sadu šablon topologií, pomocí kterých je možné automatizovat procesy pro uvedení přítomných komponent do požadovaného stavu. Další dílčí částí systému je hodnotící část, která hodnotí a kontroluje kroky uživatele pracující s danou topologií. Uživatel následně získá zpětnou vazbu na provedenou konfiguraci pro její případnou opravu.

Koncepce systému je popsána pomocí jazyka UML (Unified Modeling Language). UML je grafický modelovací jazyk pro specifikaci jednotlivých částí a funkcionalit popisovaného systému. Pomocí UML je možné systém specifikovat, vizualizovat a vytvářet dokumentaci. V současné době je vyvíjen standardizační organizací Object Management Group (OMG). Pro praktické využití má tento jazyk výhodu v jeho zjednodušeném popisu chování systému a v podpoře procesu vývoje softwaru pomocí standardních vizuálních modelů. Diagramy UML jsou rozděleny podle standardu UML 2.0 do kategorií: strukturální diagramy, diagramy chování a diagramy interakce. Pro tuto práci jsou použity některé diagramy ze zmíněných kategorií, které popisují chování systému [50].

3.5 Požadavky na systém

Po konzultaci s odpovědnými osobami pro výuku a konfiguraci síťových technologií byly definovány následující nefunkční a funkční požadavky na systém. Tyto požadavky jsou uvedeny v tabulce 3.1. Nefunkční požadavky jsou označeny NF a funkční požadavky jsou označeny FP. Pro definici priority požadavků bylo využito prioritizační metody MoSCoW [67]. Každý z požadavků je označen jedním z písmenem anglických spojení *Must have (M)*, *Should have (S)*, *Could have (C)* a *Won't have (W)*. Podle doporučení vyplývajícího z této metody by mělo být rozložení úsilí pro splnění definovaných požadavků následující:

- *Must have (M)*: Maximálně 60 % celkového plánovaného úsilí na projekt.
- *Should have (S)*: Maximálně 20 % celkového plánovaného úsilí na projekt (v součtu s *Must have* tvoří 80 % práce na projektu).
- *Could have (C)*: Zbylých 20 % tvoří *časový buffer*, pokud by vznikly nepředvídané problémy.
- *Won't have (W)*: Požadavky, u kterých bylo odsouhlaseno, že budou přesunuty do další fáze vývoje.

ID	Popis	Priorita
NP-01	Multiplatformní systém	S
NP-02	Intuitivní rozhraní	S
NP-03	Spolehlivost systému	M
NP-04	Udržitelnost systému	M
NP-05	Rozšiřitelnost systému	M
FP-01	Vytvoření nové šablony	S
FP-02	Editace šablony	S
FP-03	Vymazání šablony	S
FP-04	Zobrazení detailních informací o šabloně	S
FP-05	Zobrazení všech šablon	S
FP-06	Spuštění virtuální topologie	M
FP-07	Vypnutí virtuální topologie	C
FP-08	Otestování funkčnosti právě spuštěné topologie	M
FP-09	Spuštění virtuálního SDN kontroléru	M
FP-10	Vypnutí virtuálního SDN kontroléru	C
FP-11	Testování provedené konfigurace	M
FP-12	Zobrazení úspěšnosti při plnění praktické části	M

Tabulka 3.1: Definice funkčních a nefunkčních požadavků na systém

3.5.1 Nefunkční požadavky

Nefunkční požadavky definují, jakým způsobem bude software pracovat. Popisují, co není nezbytné pro fungování řešení, ale stále je pro projekt zásadní. Mezi základní nefunkční požadavky patří důraz na výkon, škálovatelnost, dostupnost, rozšiřitelnost, udržitelnost, spravovatelnost a bezpečnost. Při volbě nejdůležitějších požadavků pro systém je nutné počítat s tím, že se jednotlivé požadavky budou navzájem vylučovat, a proto není možné ke všem přistoupit rovnocenně [34].

- **NP-01 Multiplatformní systém** - Podmínkou pro systém je jeho funkčnost na různých typech operačních systémů (Linux, Windows a další). Systémové funkcionality by měly být konzistentní při změně podkladového operačního systému.
- **NP-02 Intuitivní rozhraní** - Systém by měl mít co nejjednodušší a nejpřehlednější uživatelské rozhraní, aby práce s ním byla pro uživatele pohodlná, efektivní a rychlá.
- **NP-03 Spolehlivost systému** - Systém by měl být spolehlivý z pohledu spuštění požadovaných operací bez nutnosti dodatečného řešení nezbytných kroků pro odstranění příčin nefunkčnosti.
- **NP-04 Udržitelnost systému** - Z důvodu udržitelnosti je nutné systém rozdělit do funkcionálních bloků, které lze spravovat samostatně.
- **NP-05 Rozšiřitelnost systému** - Z pohledu rozšiřitelnosti je důležité systém navrhnout v modulech tak, aby v případě rozšíření o nové funkcionality bylo nutné zasáhnout do stávajícího kódu minimálně.

3.5.2 Funkční požadavky

Funkční požadavky definují úlohy a služby softwaru. Definují základní činnosti, které musí být v systému implementovány pro přijímání vstupů, zpracování vstupů a generování správných výstupů. Mezi funkční požadavky patří například požadavky na výkonnost nebo design systému [33].

- **FP-01 Vytvoření nové šablony** - Systém umožní vytvořit novou šablonu pro definici virtuální topologie. Součástí procesu tvorby nové šablony bude nutné zadat jméno šablony, definovat virtuální síť, parametry SDN kontrolérů, použité testy pro zadání a volitelně další parametry.
- **FP-02 Editace šablony** - Systém umožní editovat stávající šablonu uloženou v systému. Nově vytvořená verze šablony přepíše původní verzi.
- **FP-03 Vymazání šablony** - Systém umožní smazat vybranou šablonu ze systému.
- **FP-04 Zobrazení detailních informací o šabloně** - Systém umožní vypsat detailní informace vybrané šablony.
- **FP-05 Zobrazení všech šablon** - Systém umožní vypsání všech dostupných šablon přehledně uživateli tak, aby bylo na první pohled patrné, co daná šablona nastavuje.
- **FP-06 Spuštění virtuální topologie** - Systém umožní spustit virtuální topologie podle předem připravené a vybrané šablony.
- **FP-07 Vypnutí virtuální topologie** - Systém umožní vypnout právě spuštěnou topologii.
- **FP-08 Otestování funkčnosti právě spuštěné topologie** - Systém umožní zkontrolovat funkčnost právě spuštěné virtuální topologie.
- **FP-09 Spuštění virtuálního SDN kontroléru** - Systém umožní spustit vybraný SDN kontrolér z předem definovaného seznamu kontrolérů.
- **FP-10 Vypnutí virtuálního SDN kontroléru** - Systém umožní vypnout SDN kontrolér.
- **FP-11 Testování provedené konfigurace** - Systém umožní otestovat provedenou konfiguraci uživatele v daném kroku.
- **FP-12 Zobrazení úspěšnosti při plnění praktické části** - Systém zobrazí úspěšnost uživatele během plnění kroků v praktické části zadání.

3.6 Diagramy případů užití

Diagram případů užití (Use Case Diagram) je jedním z diagramů chování definovaných v UML. Zachycuje vnější pohled na systém, čímž odhaluje hranice a rozsah funkčnosti systému. Diagram znázorňuje souvislou posloupnost souvisejících transakcí mezi účastníkem

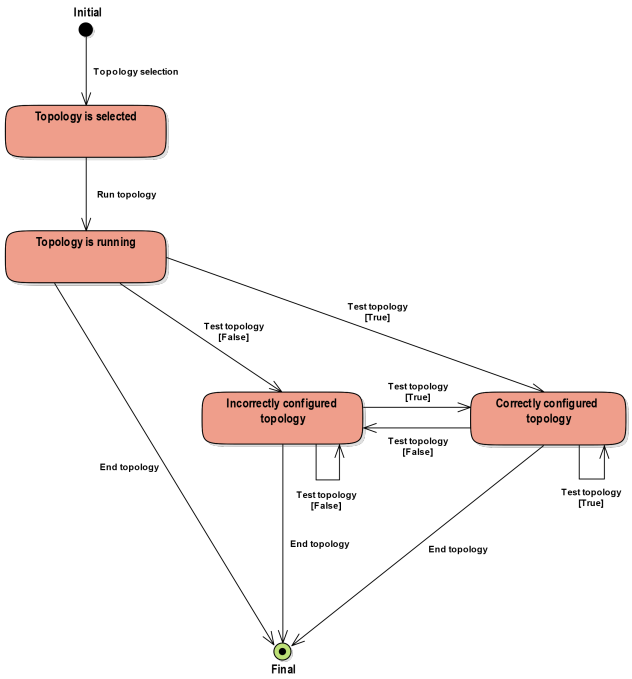
v dané roli vůči systému a systémem samotným. Prvky diagramu užití jsou účastníci a případy užití s nimi spojení. Případy užití jsou vyobrazeny oválným útvarem. Jsou to prvky posloupnosti akcí mezi účastníkem a systémem. Případy užití mohou být propojeny vazbami, které jsou definovány jako *include, exclude a generalization.* Uživatel je vyobrazen figurkou a znázorňuje externí objekt, který komunikuje se systémem. Účastníci mohou být navzájem propojeni vazbou *generalization* [16].

Z diagramů případů užití byly zpracovány diagramy pro práci s topologií sítě (viz příloha D.1), práci se šablonami (viz příloha D.2) a pro praktická cvičení (viz příloha D.3).

3.7 Stavový diagram práce s topologií

Stavový diagram zachycuje jednotlivé stavy instance objektu a přechody mezi nimi. Jedná se o způsob grafického zápisu vývoje systému v UML. Základními prvky stavového diagramu jsou stavy, přechody a události. Stav reprezentuje situaci, kdy konfigurace podmínek v systému zůstává neměnná. Tato situace nastane tehdy, když objekt čeká na vnější událost. Přechod je propojením jednotlivých stavů a je označen volitelným popiskem. Přechody mezi stavy jsou řízeny vnějším vstupem. Jsou tedy reakcí na vznik události nebo ukončení činnosti. Událost je specifikace něčeho významného, co se stane v určitém čase a prostoru.

Ve stavovém diagramu 3.1 je zachycen průběh práce s topologií. Nejprve uživatel topologii musí v grafickém rozhraní vybrat a následně ji spustit. Uživatel pokračuje konfigurací topologie podle zadání. Praktická zadání jsou definována v kapitole 4. Během práce se zadáním je uživateli zpřístupněna funkcionalita pro ohodnocení správnosti provedené konfigurace. Ukončit topologii je možné v kterémkoliv stavu topologie.



Obrázek 3.1: Stavový diagram práce s topologií

3.8 Sekvenční diagram spuštění a testování topologie

Sekvenční diagram patří do skupiny diagramů interakce. Diagramy interakce vyjadřují vnitřní chování systému a metody, které definují jednotlivé třídy. Tyto diagramy popisují skupiny objektů, které definují specifické chování systému. Sekvenční diagram popisuje chování systému v rámci konkrétního scénáře.

V diagramu jsou umístěny objekty, které mezi sebou komunikují v časové rovině, což umožňuje nahlížet na funkčnost systému v další dimenzi. Interakce mezi objekty probíhá podle definovaných kroků, kde od začátku do konce proběhne komunikace za určitý časový úsek. Samotný sekvenční diagram se skládá především z objektů a zpráv.

Objekty jsou jednotlivé třídy, které se účastní komunikace v daném diagramu. Tyto objekty jsou umístěny v horizontální rovině v horní části diagramu. Z každého objektu vede vertikálně směrem dolů přerušovaná čára, která se nazývá *dráha života objektu*. Zprávy jsou reprezentovány šipkami a jsou zasílány mezi objekty navzájem. Zprávy zleva doprava jsou označeny jako asynchronní zprávy neboli volání. Zprávy směrem zpět jsou označeny jako odpovědi nebo návratové hodnoty.

V sekvenčním diagramu v příloze D.2 je zobrazeno chování uživatele při plnění praktického zadání. Nejprve je nutné vybrat šablonu topologie, následně SDN kontrolér a volitelné parametry pro danou relaci. Po výběru požadovaných parametrů je nutné stisknout tlačítko pro spuštění topologie. Následně proběhne spuštění SDN kontroléru a virtuální sítě. Uživatel je informován o stavu spouštěných komponent topologie.

V druhé části diagramu je zobrazeno vyhodnocení konfigurace, které uživatel provedl. Pro vyhodnocení je nutné stisknout tlačítko *Test Topology*, pomocí kterého se začnou pomocí třídy *TestExecutor* spouštět jednotlivé testy definované v šabloně topologie. Uživatel je informován o stavu testování topologie formou zobrazení výsledků v hlavním okně GUI.

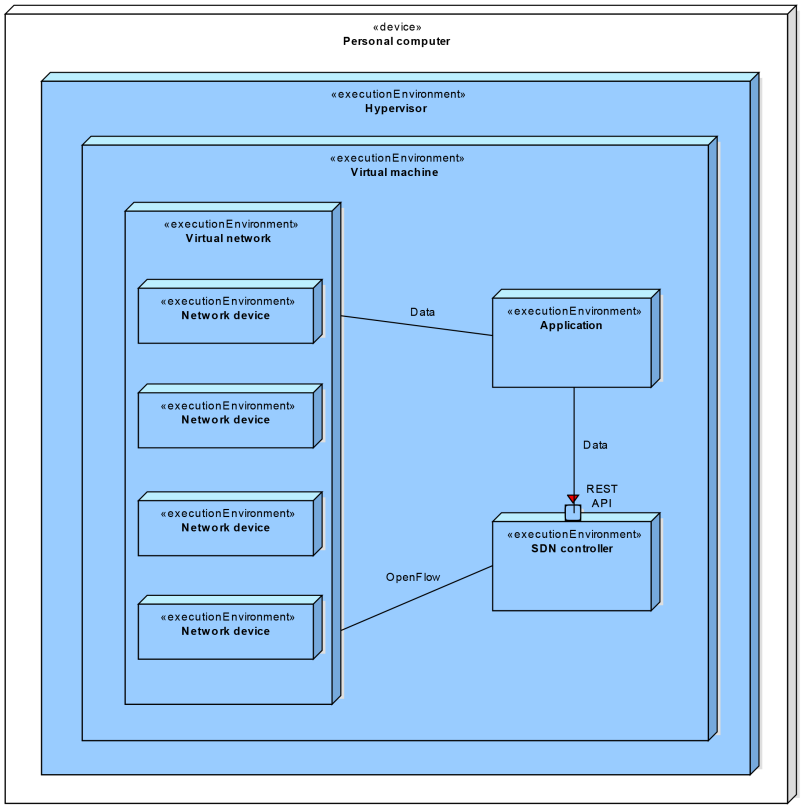
3.9 Diagram tříd systému

Diagram tříd slouží pro popis statické struktury systému a znázorňuje operace a souvislosti mezi objekty. Datové struktury jsou zařazeny do tříd. Vztahy těchto tříd jsou následně zobrazeny pomocí čar zakončených konkrétním symbolem dle definice a popiskem. Vztahy mezi objekty v diagramu mohou být na úrovni instance případně na úrovni třídy. Diagram tříd je základním diagramem pro objektově orientované modelování. Třídy mohou být doplněny definicí svých metod, funkcí a parametrů uchovávajících pro správnou ;funkčnost systému.

V diagramu tříd v příloze D.3 je zobrazena komunikace mezi nejdůležitějšími třídami v projektu. Základ systému je postaven na komponentách *model-view-controller*. Třída model odkazuje na několik důležitých datových struktur pro systém. *SDNController* je abstraktní třída, která definuje rozhraní pro komunikaci s SDN kontrolérem. Třída *VirtualNetwork* je také abstraktní třídou a definuje rozhraní pro komunikaci s jejími implementacemi. Další důležitou třídou je *TextExecutor*, která tvoří exekuční prostředí pro vykonávání testů definovaných v šabloně topologie. Jednotlivé testy jsou definovány pomocí struktury testů znázorněných v diagramu. Třída model současně importuje šablony topologií a definici virtuálních sítí podle definovaných praktických zadání.

3.10 Diagram nasazení systému

Diagram nasazení zobrazuje způsob implementace automatizovaného systému. Jedná se o přiřazení jednotlivých SW prvků (artefaktů) na výpočetní jednotky (uzly). Klasický diagram nasazení modeluje obecné řešení problematiky nasazení software. Konkrétní diagram nasazení je zobrazuje řešení pro konkrétně definovaný případ [3].



Obrázek 3.2: Diagram nasazení systému

Na obrázku 3.2 je zobrazen konkrétní diagram nasazení automatizovaného systému. V systému je přítomno běhového prostředí, kterým je operační systém osobního počítače. V operačním systému je se nachází uvedený hypervisor, což je komponenta, která umožňuje spouštění virtuálních strojů a současně jim přiděluje prostředky pro jejich funkci. V hypervisoru je zobrazen virtuální stroj, který se obsahuje virtuální síť, která představuje sadu několika podprogramů simulujících fyzické síťové prvky. Tato virtuální síť komunikuje pomocí OpenFlow protokolu s SDN kontrolérem. Obě zmíněné části, jak virtuální síť, tak SDN kontrolér, řídí automatizovaný systém označený v diagramu jako aplikace.

Hypervisor pracuje s HW prostředky hostitelského osobního počítače. Osobní počítač současně může komunikovat s okolním prostředím, což dodává celému řešení možnost růstu pro další rozšíření.

3.11 Hierarchický vícevrstvý model

Následující definice pro hierarchický vícevrstvý model je převzata ze zdroje [35], pokud není uvedeno jinak.

Vícevrstvý přístup pro modelování komplexních systémů zahrnuje dvě základní oblasti:

- vícevrstvé modely (formální reprezentace architektury)
- referenční modely (definice vrstev)

Typem vícevrstvé sítě je hierarchická vícevrstvá síť, která je ve své spodní vrstvě tvořena fyzickou sítí, a ostatní vrstvy jsou virtuální. Tyto virtuální vrstvy jsou úzce svázané s fyzickou topologií. Koncept hierarchických sítí je založen na dvou základních faktech, kterými jsou:

- pro každý uzel na dané vrstvě existuje odpovídající uzel nebo uzly na vrstvě o úroveň níže
- pro každou cestu mezi dvěma uzly na dané vrstvě existuje cesta nebo cesty mezi odpovídajícími uzly na vrstvě u úroveň níže

Na základě faktů popsaných výše můžeme definovat testovaný systém SUT (System Under Test) jako vícevrstvou projekci sítě:

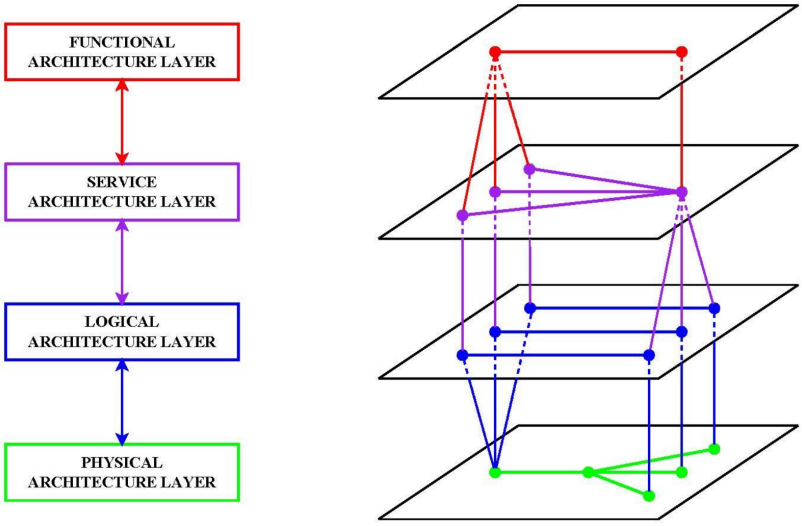
$$M = (V, E) \tag{3.1}$$

kde M je vícevrstvý 3D graf (viz obrázek 3.3) odvozený od specifikace SUT; V(M) ke konečná, neprázdná množina komponent SUT; E(M) je konečná, neprázdná množina vzájemného propojení komponent. Následně definujeme:

$$v = \bigcup_{\alpha=1}^{L} V^{\alpha} \tag{3.2}$$

$$E = \left(\bigcup_{\alpha=1}^{L} E^{\alpha} \right) \bigcup \left(\bigcup_{\alpha=2}^{L} E^{\alpha,(\alpha-1)} \right) \tag{3.3}$$

kde V^{α} je konečná, neprázdná množina komponent SUT na vrstvě α; E^{α} je konečná, neprázdná množina vzájemného propojení komponent na vrstvě α; $E^{\alpha,(\alpha-1)}$ je konečná, neprázdná množina projekcí mezi komponenty na vrstvě α a $\alpha - 1$; L je počet vrstev SUT, $1 \leq \alpha \leq L$.

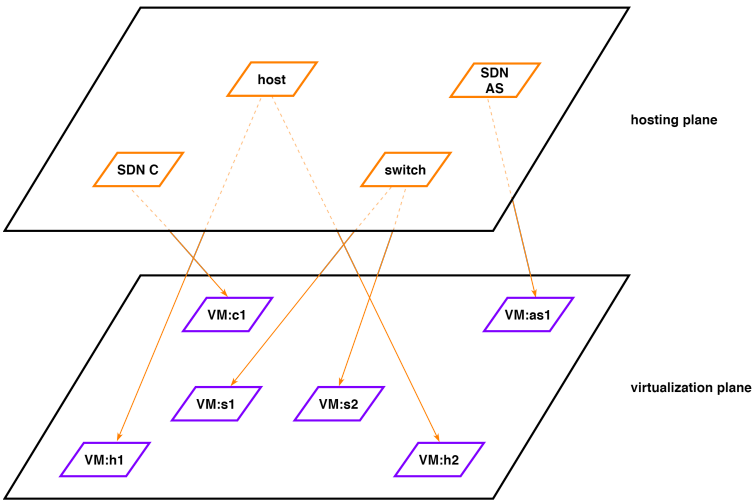


Obrázek 3.3: Čtyřvrstvý model distribuovaného systému [36]

3.11.1 Rozšíření vícevrstvého modelu

Základní model distribuovaného systému (viz obrázek 3.3) popisuje v několika vrstvách vztah mezi HW a SW pro konkrétní aplikaci. Pro reprezentaci komunikace závislých komponent použitých při návrhu automatizovaného systému byl rozšířen vícevrstvý model pro popis distribuovaného systému o zobrazení virtualizované topologie.

Popis funkce automatizovaného systému je rozdělen do dvou grafů, kde první z nich (viz obrázek 3.4) popisuje vztah *hosting plane* s *virtualization plane*. Vztah mezi těmito vrstvami je založen na vzniku virtuálních instancí komponent, které jsou definovány v šabloně topologie (viz příloha E.1.1). Druhý graf se zaměřuje na popis virtuální infrastruktury (viz obrázek 3.5).



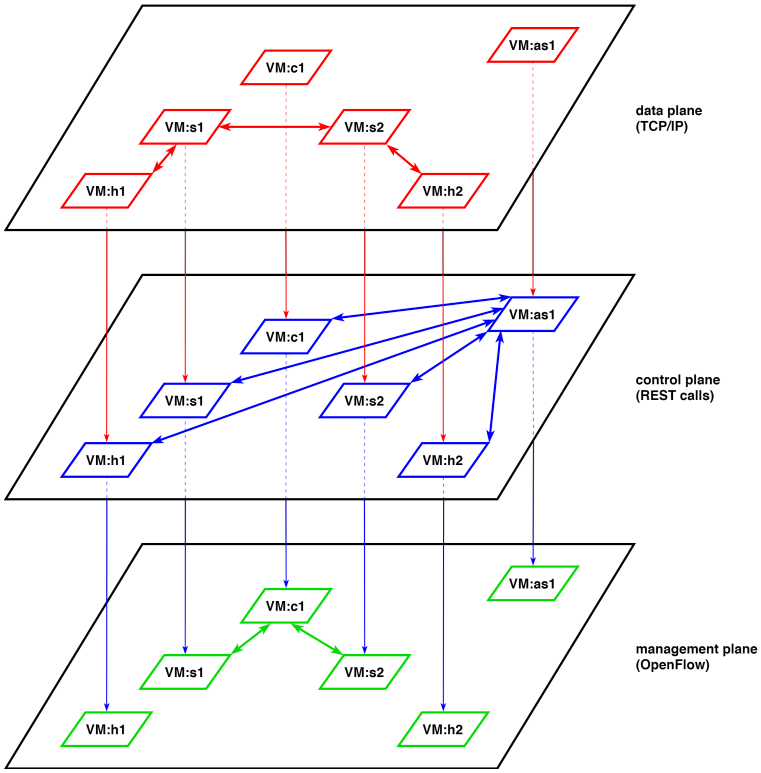
Obrázek 3.4: Hierarchický vícevrstvý model

Za předpokladu identické množiny uzlů na všech vrstvách grafu a implicitně definované struktury mezivrstev, lze definovat *multiplex network* jako vícevrstvou síť, kterou tvoří množina grafů uvnitř dané vrstvy, tj.:

$$M = \bigcup_{\alpha=1}^{L} G^{\alpha} \tag{3.4}$$

kde $V^{\alpha} = V^{\beta} = V(M)$ pro $1 \leq \alpha \neq \beta \leq L$; G^{α} je graf na dané vrstvě $G^{\alpha} = (V^{\alpha}, E^{\alpha})$.

Pro další analýzu si můžeme virtuální strukturu představit jako *multiplex network* (viz obrázek 3.5). Klíčovou vlastností *multiplex network* je stejný počet uzlů na všech vrstvách modelu. Tento typ modelu byl vytvořen pro použití k analýze sociálních sítí [41], [77], [58].



Obrázek 3.5: Virtuální infrastruktura

Vrstva *data plane* obsahuje virtuální instance použitých nástrojů, které mezi sebou komunikují a přenáší data pomocí TCP/IP. Na této vrstvě spolu komunikují koncové stanice a síťová zařízení, která jsou v tomto případě přepínače.

Na následující vrstvě *control plane* je zobrazena komunikace mezi všemi přítomnými instancemi a automatizovaným systémem. Komunikace probíhá především pomocí REST API volání.

Na vrstvě *m anagement plane* je uvedena komunikace protokolu OpenFlow (viz kapitola 2.4). Této komunikace se účastní SDN kontrolér a síťová zařízení, která jsou k němu registrována.

3.12 Výběr a porovnání vhodných nástrojů pro implementaci

3.12.1 HW a virtuální přepínače

Z pohledu výběru vhodného přepínače pro tuto diplomovou práci, je nutné zaměřit se na snadnou práci s přepínačem a jaké jsou možnosti jeho konfigurace. Nelze porovnávat HW přepínače s virtuálními. Pro HW přepínače je značnou výhodou možnost běhu operačního systému na vyhrazeném HW. U virtuálních přepínačů jsou HW prostředky sdíleny s dalšími procesy běžícími na hostitelském stroji. Dalším pohledem, který je nutné zvážit, jsou náklady na realizaci vybraného řešení. Z pohledu virtuálních přepínačů je možné provozovat virtuální síť tak velkou, kolik nám dovolí výpočetní výkon hostitelského stroje. U HW přepínačů je nutné při potřebě zvětšení topologie pořídit další síťová zařízení.

Cisco HW přepínače

Cisco přepínače pracují ve dvou základních režimech, kde jedním z nich je klasický režim, který je dnes ještě běžně používán. V tomto režimu má každý přepínač svou datovou i řídicí rovinu. Druhou možností komunikace je režim s podporou protokolu OpenFlow. Pro možnost napojení přepínače na SDN kontrolér je nutné přepínač patřičně nakonfigurovat.

Pro starší přepínače řady Catalyst se tento doplněk nazývá *Cisco Plug-in* pro OpenFlow. Od vydání operačního systému Cisco NX-OS verze 7.0(3)I5(1) je nahrazován novou technologií pojmenovanou *OpenFlow Agent*. OpenFlow Agent je plně podporován řadou Nexus 9000. Pro spuštění například na přepínačích řady Nexus 3000 je možné zadat příkaz **system switch-mode n9k**, a tím přejít do režimu kompatibility pro vyšší řadu. Pro samotnou aktivaci protokolu OpenFlow slouží příkaz **feature openflow**.

Přepínače Cisco podporují pouze verze 1.0 a 1.3 OpenFlow protokolu. V tabulce 3.2 jsou uvedeny verze OpenFlow protokolu podporované s jednotlivými SDN kontroléry [11].

OpenFlow verze	**Podporované SDN kontroléry**
OpenFlow 1.0	Cisco Open SDN Controller nebo POX controller.
OpenFlow 1.3	Cisco Open SDN Controller, Ixia, OpenDaylight nebo Ryu

Tabulka 3.2: Podporované verze OpenFlow protokolu s jednotlivými SDN kontroléry [11]

Další možností, jak komunikovat s SDN přepínači, která Ciscem preferována, je napojení přepínačů na orchestrátor sítě DNA Center (viz kapitola 3.12.2).

Open-source virtuální přepínače

Výhodou virtuálních přepínačů je jejich snadná konfigurace, úpravy logické struktury a spuštění libovolného množství v jednou virtualizovaném prostředí. Nevýhodou může být stabilita závislá na podkladovém operačním systému, případně nedostupná dokumentace. Topologii virtualizované sítě je možné za běhu měnit pomocí CLI případně GUI, pokud je dostupné.

V tabulce 2.1 je uveden výběr důležitých open-source přepínačů z pohledu jejich vývoje, funkcionality a udržování dokumentace.

Software switch	Commits	Releases	Forks	Language	Last activity
OpenWrt	46698	25	3700	C	2Q 2020
Open vSwitch	18017	111	1400	C	2Q 2020
Snabb	9456	93	295	Lua	2Q 2020
VPP	8349	79	209	C	2Q 2020
BESS	2890	3	107	C++	2Q 2020
bmv2	241	0	52	C	1Q 2018
Indigo	1230	1	29	C	4Q 2015
LINC	1132	5	78	Erlang	3Q 2015

Tabulka 3.3: Přehled vývoje open-source SDN přepínačů (aktualizace k 9.5.2020)

Nad následujícími přepínači bylo uvažováno pro výběr vhodného řešení pro tuto diplomovou práci.

OpenWrt

Projekt OpenWrt je operační systém Linux upravený pro *embedded* zařízení. Místo vytvoření jedné statické aplikace se projekt OpenWrt zaměřil na tvorbu systému založeného na správě balíčků, což umožňuje volbu aplikací a konfigurace pro konkrétního výrobce. Dovoluje upravit zařízení pro potřeby dané aplikace. Pro vývojáře je velkou výhodou možnost sestavení aplikace bez potřeby sestavit celý firmware pro zařízení. Typickým zařízením pro tento *framework* jsou bezdrátové směrovače [52].

Open vSwitch

Open vSwitch je multivrstvý softwarový přepínač. Cílem tohoto nástroje je implementovat přepínač produkční kvality, který podporuje standardní správu rozhraní. Umožňuje funkcionalitu pro odesílání dat doplnit o programovou správu. Velkou výhodou je podpora virtualizace a umožnění správy rozhraní napříč virtuálními síťovými vrstvami. Přepínač je napsaný v platformě nezávislém programovacím jazyce C [75].

Snabb

Snabb je open-source sada nástrojů umožňující virtualizovat ethernetovou vrstvu a její služby. Díky podpoře virtualizace je možné manipulovat s rámci v přepínači. Snabb podporuje skriptovací jazyk Lua. Součástí sady nástrojů je například filtr paketů nebo generátor provozu [70].

VPP

VPP (Vector Packet Processing) je síťová datová rovina jako součást projektu Linux Foundation FD.io. Je navržen na vrcholu vývojové sady pro datovou rovinu (DPDK), aby provozoval síťové pracovní zátěže při vysokých rychlostech pomocí technologie zpracování vektorových paketů. Vektorový přístup k paketům, oproti skalárnímu způsobu zpracování, umožňuje zpracovat několik paketů v jednom časovém okamžiku [22].

BESS

BESS (známý jako SoftNIC) je modulární framework pro softwarové přepínače. Umožňuje nakonfigurovat manuálně nebo pomocí kontroléru způsob zpracování paketu pomocí modulů integrovaných ve frameworku. BESS vznikl díky spolupráce institucí University of California, Berkeley a Nefeli Networks. BESS je zveřejněn pod licencí BSD a jedná se o open-source nástroj. Současně se jedná o první softwarový přepínač, který je navržen pro podporu NFV [48].

bmv2

Bmv2 (behavioral model version 2) je druhou verzí referenčního P4 softwarového přepínače. Tato druhá verze je napsaná v jazyce C++11. Jako vstup je požadován JSON soubor generovaný z P4 programu pomocí P4 kompilátoru. Mbv2 není vytvořen pro fungování na produkční úrovni, jedná se pouze o nástroj pro vývoj, testování a ladění výkonu pro P4 datové a řídicí roviny přepínačů [55].

Indigo

IVS (Indigo Virtual Switch) je OpenFlow virtuální přepínač navržený pro vysoký výkon a minimální požadavky na administraci. Je postaven na Indigo platformě, která poskytuje společné jádro pro celou řadu fyzických a virtuálních přepínačů [60].

LINC

LINC (Link Is Not Closed) je OpenFlow software napsaný v jazyce Erlang. V operačním systému je integrován jako Erlang node v prostředí uživatele. Tato integrace sice není velmi efektivní, co se týče výkonu, ale umožňuje vysokou flexibilitu a dovoluje rychlý vývoj a testování nových OpenFlow funkcionalit [24].

3.12.2 SDN kontroléry

Pro vhodný výběr SDN kontrolérů je nutné zamyslet se mnohem více nad parametry, než jen nad dostatečným výkonem pro výpočty topologie. Celá řada kontrolérů má svůj vývoj rozprostřený přes několik komunit, které svými silami udržují a rozšiřují open-source repozitáře. Oproti tomu stojí komerční SDN kontroléry, které díky vysokým potřebám dnes ve velké míře dnes běží na vyhrazeném serveru. V současné době mnoho komerčních kontrolérů není k dispozici pro nasazení ve virtuální podobě. Oproti tomu open-source kontroléry jsou pro virtualizaci uzpůsobené.

Z pohledu práce s SDN kontroléry a budoucího rozšíření nástrojů používaných pro správu sítě je nutné zamyslet se nad rozhraními, která zvažujeme v nejbližší době používat. Komunikace s kontroléry je většinou možná pomocí severního rozhraní, případně východního nebo západního. Jižní rozhraní slouží pro komunikaci se síťovými prvky. Pro severní rozhraní je nejčastěji k dispozici JAVA API nebo REST API rozhraní. Některé kontroléry umožňují konfiguraci pomocí RESTCONF případně NETCONF.

Další vlastnosti, které je vhodné neopomenout, jsou možnosti rozšiřitelnosti systému, *upgradu* na novou verzi, *stackování* více kontrolérů, možnost zálohy a obnovení nastavení, případně počet a typy podporovaných síťových zařízeních pro správu OpenFlow protokolem. SDN kontroléry podporují určité verze OpenFlow protokolu od 1.0 až po 1.5, kde poslední verze je v současné době ve velké míře v SDN kontrolérech označena jako experimentální stav.

Některé kontroléry je možné spravovat pomocí interaktivního CLI, jiné pomocí grafického rozhraní v HTML případně v jiném PHP *frameworku*. Některé kontroléry neobsahují grafický modul, tudíž je s nimi možné komunikovat pouze pomocí API.

Controller	**Commits**	**Releases**	**Forks**	**Language**	**Last activity**
ONOS	14689	178	598	Java	2Q 2020
OpenDaylight	8601	70	315	Java	2Q 2020
Ryu	3258	109	903	Python	2Q 2020
Floodlight	3146	10	561	Java	1Q 2020
POX	1826	1	427	Python	4Q 2019
Trema	3346	70	74	Gherkin	3Q 2018
NOX	28	0	83	C++	1Q 2014

Tabulka 3.4: Přehled vývoje open-source SDN kontrolérů (aktualizace k 9.5.2020)

Před instalací a spuštěním open-source kontrolérů, které vyžadují Javu, je nutné ji nejprve nainstalovat. Instalace a nastavení potřebných atributů pro Java 1.8 je uvedena v instalačním skriptu 3.1.

```
# aktualizace systemu
sudo apt update
sudo apt upgrade

# instalace openjdk-8-jdk
sudo apt install openjdk-8-jdk

# nastaveni systemovych atributu a~overeni funkcnosti
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
echo $JAVA_HOME
export PATH=$PATH:$JAVA_HOME/bin
echo $PATH
java -version
```

Seznam zdrojových kódů 3.1: Instalace Java 1.8

DNA Center

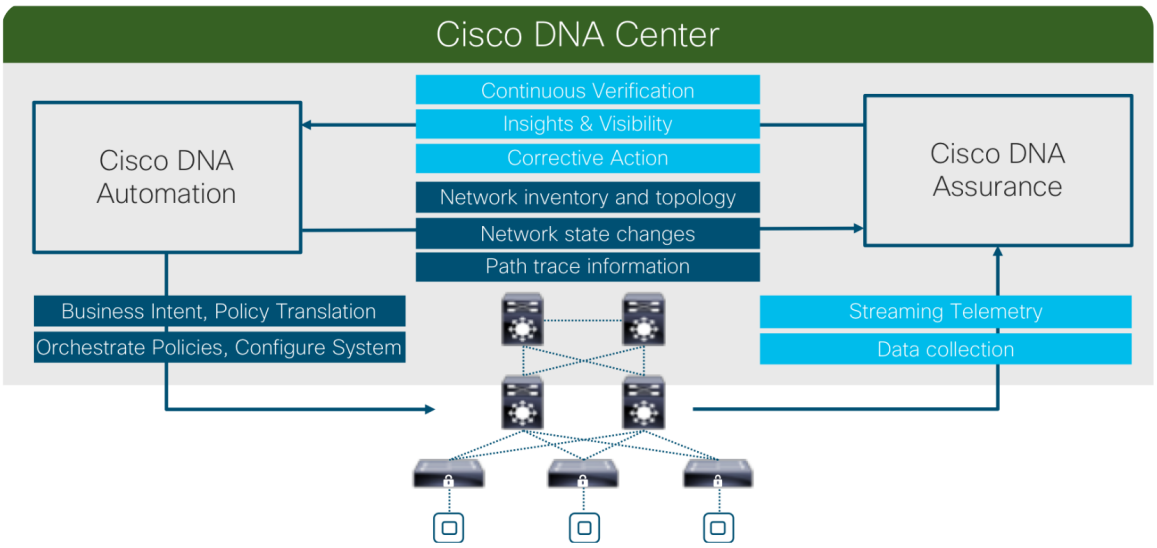
Cisco DNA Center je samotným jádrem síťové architektury zvané *intent-based*. Tato architektura se zaměřuje na převedení marketingových a procesních záměrů do síťové topologie a jednoduchosti nastavení jednotlivých cílů. DNAC poskytuje nástroje pro automatizaci obchodních záměrů. Součástí jsou i analytické a podpůrné nástroje pro komplexní přehled o funkčnosti sítě prostřednictvím dat a statistik.

Pro komunikaci s DNA Center jsou k dispozici 4 základní API [12]:

- *Intent API* - Severní REST API rozhraní pro komunikaci s aplikací třetí strany. Pro komunikaci používá HTTPS metody (GET, POST, PUT, DELETE).
- *Integration API* - Západní rozhraní a mechanismy pro integraci servisní aplikace a podpory třetích stran.
- *Events and Notifications Services* - Východní rozhraní pro poskytnutí třetím stranám možnosti reakce na události z DNAC.
- *Multivendor SDK* - Jižní rozhraní pro komunikaci se síťovými zařízeními třetích stran. Umožňuje generovat balíčky pro potřeby jižních rozhraních ostatních zařízeních.

DNAC je možné virtualizovat v nástroji poskytovaném společností Cisco, který je označen *DNA Center Always-on Sandbox*. Jedná se o nástroj, kdy je instance DNAC spuštěna jako demo verze, ke které je možné se přihlásit a otestovat možnosti kontroléru. Přihlašovací údaje pro tuto alternativu jsou *devnetuser : Cisco123!*. Druhou možností je využít *Reservation-based DevNet sandbox*, který má již předinstalované různé balíčky vhodné pro vývoj externích aplikací. Třetí a poslední možností je vlastnit DNAC, tedy HW server umístěný v interní síti.

Na obrázku 3.6 je znázorněna architektura kontroléru DNA Center. Na tomto obrázku je patrné, že kontrolér obsahuje několik vzájemně integrovaných komponent. Jsou to komponenty pro orchestraci sítě, automatizaci a zajištění funkčnosti sítě.



Obrázek 3.6: Architektura kontroléru DNA Center [38]

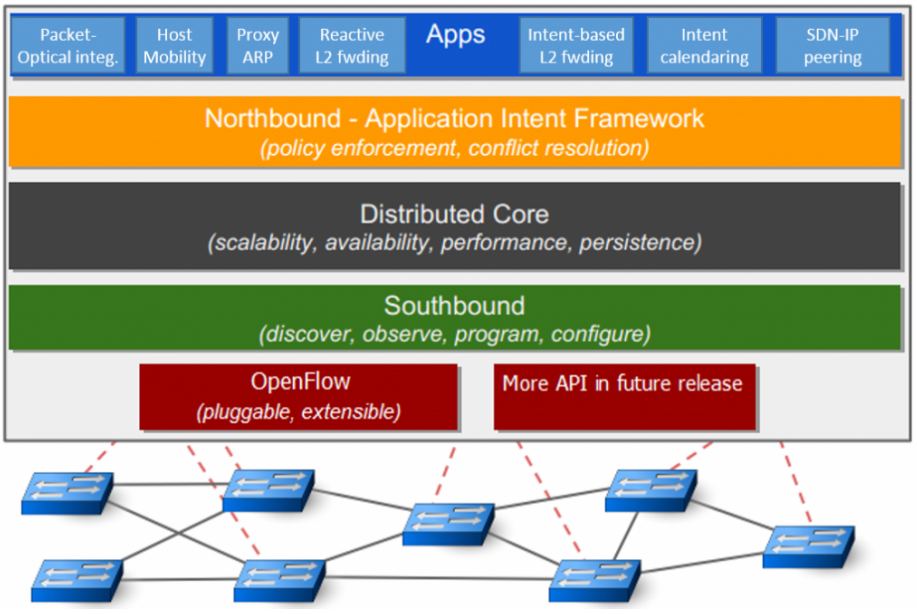
ONOS

Programovací jazyk	Java
Testovaná verze kontroléru	1.6.0, 1.15.0, 2.0.0, 2.1.0, 2.2.1, 2.2.2, 2.3.0
Grafické rozhraní	ano
API	Java API, REST API
Nejvyšší podporovaná verze OpenFlow protokolu	OpenFlow 1.5
Odkaz na dokumentaci	wiki.onosproject.org

Tabulka 3.5: Vlastnosti SDN kontroléru ONOS

ONOS je kontrolér, který má ze všech open-source kontrolérů největší aktivitu ve vývoji a poskytuje celou řadu funkcionalit pro správu sítě. Pro komunikaci s kontrolérem je vystaveno REST API. Mimo OpenFlow podporuje celou řadu dalších protokolů pro komunikaci se síťovými zařízeními. Vyvíjí se pomocí nástroje `Bazel`, což je vývojářský nástroj od společnosti Google. Kontrolér je napsaný v jazyce Java a běží v JVM (Java Virtual Machine). Pro správu funkcí používá OSGi (Open Services Gateway initiative), který je implementovaný pomocí programu Karaf.

ONOS má modulární architekturu složenou z několika vrstev (viz obrázek 3.7). Aplikační vrstva je nejvyšším stupněm abstrakce. Síťová vrstva je používána aplikacemi pro správu topologie.



Obrázek 3.7: Architektura kontroléru ONOS [73]

V tabulce 3.5 je uveden seznam testovaných verzí kontroléru ONOS. Po testování několika verzí se ukázala jako jediná funkční verze nevykazující problémy 1.15.0. Ostatní verze vykazovaly problémy se systémem Karaf používající OSGi.

```
# vytvoreni adresare pro ONOS
sudo mkdir /opt
cd /opt

# stazeni kontroleru v dane verzi ONOS_VERSION
sudo wget -c http://downloads.onosproject.org/release/onos-$ONOS_VERSION.tar
    .gz

# rozbaleni archivu do adresare /opt
sudo tar -xzf onos-$ONOS_VERSION.tar.gz

# prejmenovani adresare na "onos"
sudo mv onos-$ONOS_VERSION onos

# spusteni a otestovani funkcnosti
export TERM=xterm-color
/opt/onos/bin/onos-service start
```

Seznam zdrojových kódů 3.2: ONOS instalace a spuštění

OpenDaylight

Programovací jazyk	Java
Testované verze kontroléru	Oxygen, Sodium, Carbon, Magnesium
Grafické rozhraní	ano
API	NETCONF, RESTCONF
Nejvyšší podporovaná verze OpenFlow protokolu	OpenFlow 1.0/1.3
Odkaz na dokumentaci	docs.opendaylight.org

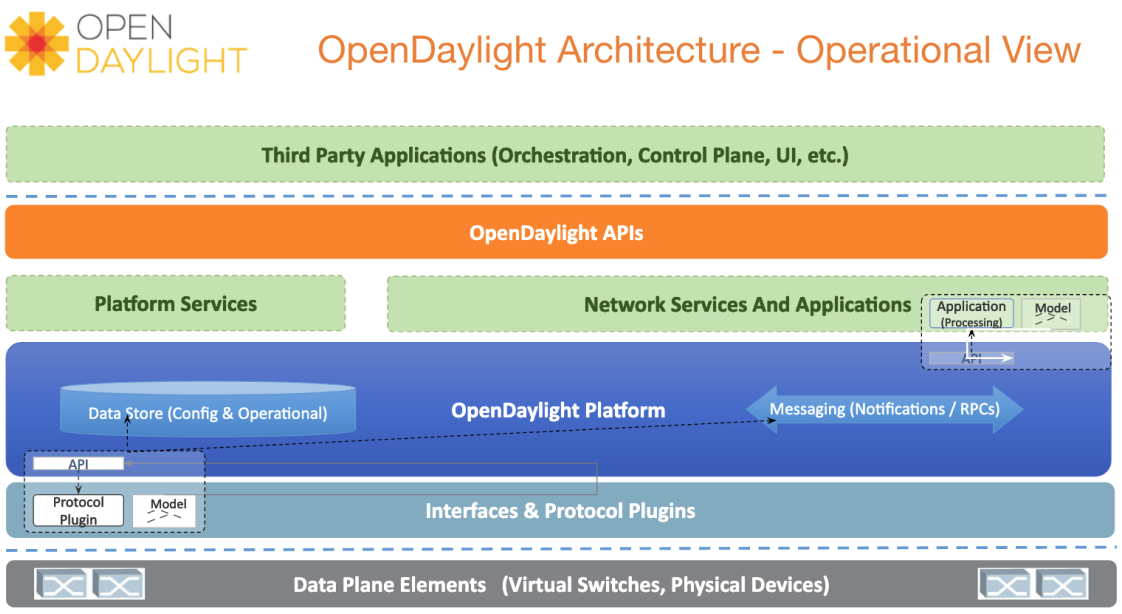
Tabulka 3.6: Vlastnosti SDN kontroléru OpenDaylight

OpenDaylight je sestavený z komponent, které je možné přidávat a odebírat pomocí nástroje `Karaf`, který poskytuje běhové prostředí systému. Je napsaný v programovacím jazyce Java a běží v JVM. Podporuje protokoly RESTCONF, NETCONF v severním rozhraní a pro jižní rozhraní podporuje OVSDB a OpenFlow. Grafické rozhraní kontroléru zajišťuje rozhraní DLUX. S kontrolérem je možné komunikovat skrze REST API. Kontrolér podporuje celou řadu rozšíření, například integraci s OpenStack, jazyk P4 nebo podporu virtualizace za pomocí kontejnerů. Architektura kontroléru je zobrazena na obrázku 3.8.

```
# instalace prerekvizit
sudo apt-get install software-properties-common
sudo apt-get install maven git unzip

# stazeni kontroleru v dane verzi ODL_VERSION
wget https://nexus.opendaylight.org/content/repositories/public/org/
    opendaylight/integration/distribution-karaf/ODL_VERSION/distribution-
    karaf-ODL_VERSION.zip
```

```
7
8  # rozbaleni stazeneho adresare
9  unzip distribution-karaf-ODL_VERSION.zip
10
11 # presun do rozbaleneho adresare a spusteni kontroleru
12 export TERM=xterm-color
13 cd distribution-karaf-ODL_VERSION
14 ./bin/karaf
15
16 # instalace balicku pro podporu DLUX grafickeho rozhrani a pro praci s
       OpenFlow protokolem
17 feature:install odl-dluxapps-aplications odl-restconf odl-l2switch-switch-ui
        odl-mdsal-apidocs odl-openflowplugin-all
18
19 # instalace OpenDaylight-Openflow-App pro vizualizaci OpenFlow topologie a
       nastroje Grunt pro spusteni weboveho serveru Node.js
20 sudo apt-get instsall npm
21 sudo npm update -g npm
22 sudo npm install -g grunt-cli
23 git clone  git://github.com/CiscoDevNet/OpenDaylight-Openflow-App.git
24
25 # spusteni weboveho serveru Node.js dostupneho na adrese http://localhost
       :9000
26 cd OpenDaylight-Openflow-App
27 grunt
```

Seznam zdrojových kódů 3.3: OpenDaylight instalace a spuštění



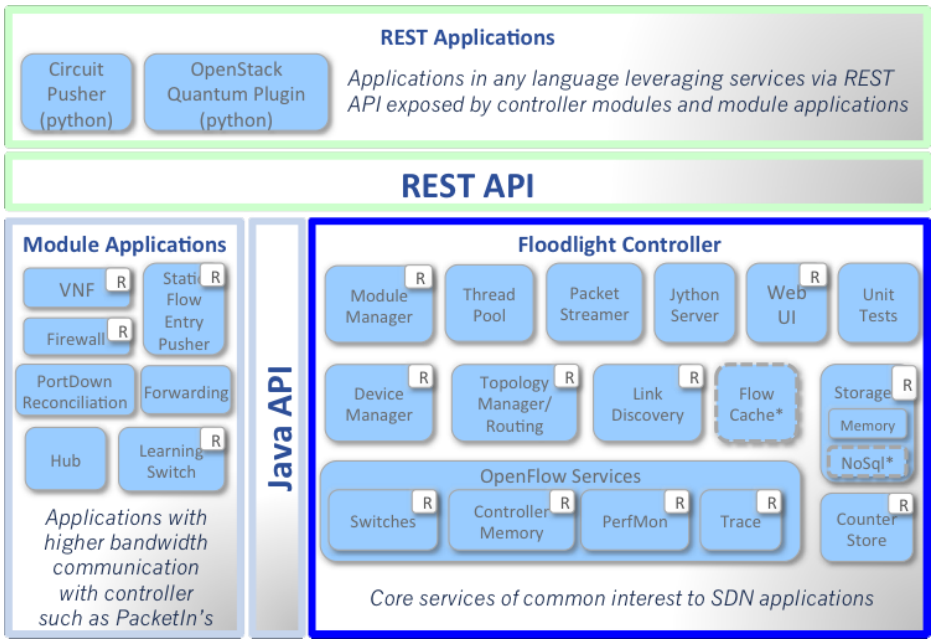
Obrázek 3.8: Architektura kontroléru OpenDaylight [51]

Floodlight

Programovací jazyk	Java
Testovaná verze kontroléru	1.2
Grafické rozhraní	ano
API	REST API
Nejvyšší podporovaná verze OpenFlow protokolu	OpenFlow 1.5
Odkaz na dokumentaci	floodlight.atlassian.net/wiki/spaces/... .../floodlightcontroller/overview

Tabulka 3.7: Vlastnosti SDN kontroléru Floodlight

Floodlight podporuje všechny verze OpenFlow protokolu včetně poslední verze 1.5. Je napsaný v jazyce Java a současně obsahuje vlastní GUI. Pro komunikaci s kontrolérem je dostupné REST API. Architektura kontroléru Floodlight je znázorněna na obrázku 3.9. Kontrolér má modulární architekturu, která obsahuje komponenty pro správu topologie a zařízení, výpočty cest, grafické rozhraní a další.



Obrázek 3.9: Architektura kontroléru Floodlight [79]

```
# instalace prerekvizit
sudo apt-get install build-essential ant maven python-dev

# instalace kontroleru ze zdrojoveho kodu
git clone git://github.com/floodlight/floodlight.git
cd floodlight
git submodule init
```

```
8  git submodule update
9  ant
10
11 # spusteni kontroleru
12 java -jar ./target/floodlight.jar
```

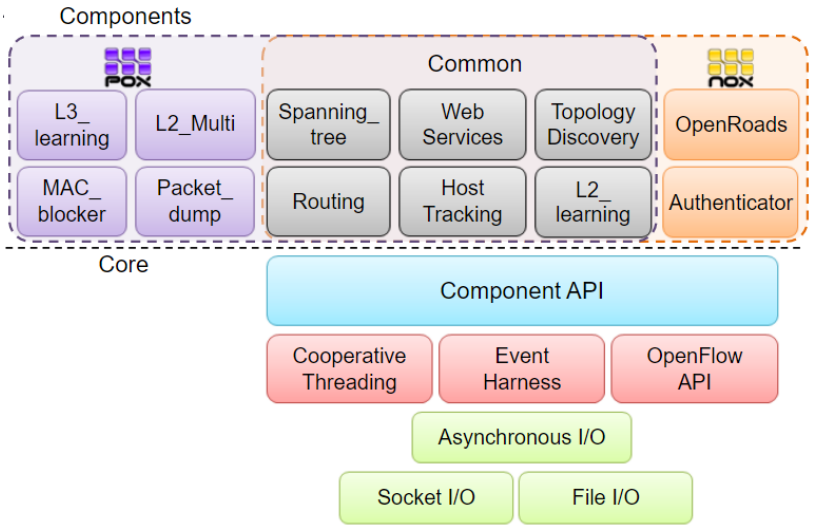
Seznam zdrojových kódů 3.4: Floodlight instalace a spuštění

POX

Programovací jazyk	Python
Testovaná verze kontroléru	eel
Grafické rozhraní	ano
API	JSON-RPC, REST API
Nejvyšší podporovaná verze OpenFlow protokolu	OpenFlow 1.0
Odkaz na dokumentaci	https://noxrepo.github.io/pox-doc/html/

Tabulka 3.8: Vlastnosti SDN kontroléru Pox

NOX byl vyvinut společností Nicra a v roce 2008 byl licenčně upraven jako open-source. POX je novou verzí původního NOX kontroléru a poskytuje framework pro komunikaci s SDN přepínači pomocí OpenFlow a OVSDB protokolů. Nově je tato vývojová větev založena na jazyce Python. Kontrolér podporuje virtualizaci. Podporuje pouze verzi OpenFlow 1.0. Pro komunikaci poskytuje REST API. Neobsahuje ovšem grafické rozhraní.



Obrázek 3.10: Srovnání architektur kontrolérů POX a NOX [65]

Na obrázku 3.10 je zobrazeno porovnání architektur kontrolérů NOX a POX a jejich rozdíly v implementaci jednotlivých funkcionalit.

```
# prerekvizity pro kompatibilitu s Python 3 pomoci PyPy
sudo apt install snapd
sudo snap install pypy --classic

# instalace kontroleru ze zdrojoveho kodu
git clone http://github.com/noxrepo/pox
cd pox
git checkout eel

# spusteni kontroleru s aplikaci "forwarding.l2_learning"
./pox.py forwarding.l2_learning

# stazeni a instalace nastroje Gephi pro graficke zobrazeni OpenFlow topologie
wget https://github.com/gephi/gephi/releases/download/v0.9.2/gephi-0.9.2-linux.tar.gz

# rozbaleni archivu
sudo tar -xzf gephi-0.9.2-linux.tar.gz

# spusteni programu Gephi
cd gephi-0.9.2
./bin/gephi
```

Seznam zdrojových kódů 3.5: POX instalace a spuštění

Ryu

Programovací jazyk	Python
Testovaná verze kontroléru	4.34
Grafické rozhraní	ano
API	REST API, Application API
Nejvyšší podporovaná verze OpenFlow protokolu	OpenFlow 1.5
Odkaz na dokumentaci	ryu.readthedocs.io

Tabulka 3.9: Vlastnosti SDN kontroléru Ryu

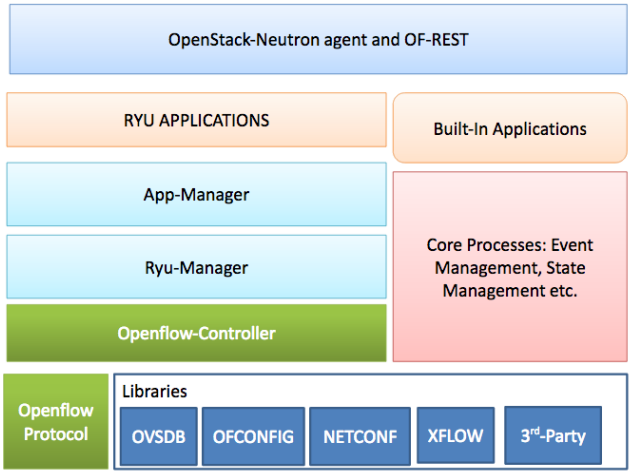
RYU je kontrolér, který podporuje protokol OpenFlow nejvyšší verze 1.5. Je napsaný v Pyhonu a poskytuje grafické rozhraní. Jedná se kontrolér, který je složený ze stavebních síťových bloků. Poskytuje rozhraní API pro komunikaci a nastavení kontroléru. Mimo protokolu OpenFlow podporuje i protokoly NETCONF, OF-Config a další.

Ryu kontrolér umí pracovat s protokolem OpenFlow, zpracovávat asynchronní zprávy a analyzovat příchozí pakety. Na obrázku 3.11 je zobrazena architektura tohoto kontroléru.

```
# prerekvizity pro kontroler Ryu (Ubuntu 16.04 LTS nebo novejsi)
apt install gcc python-dev libffi-dev libssl-dev libxml2-dev libxslt1-dev zlib1g-dev

# instalace z python balicku
```

```
pip install ryu

# instalace ze zdrojoveho kodu
git clone git://github.com/osrg/ryu.git
cd ryu
sudo python ./setup.py install

# spusteni a otestovani funkcnosti kontroleru
ryu-manager --version

# volitelne pozadavky (OF-Config, NETCONF, BGP, Zebra protocol, ...)
pip install -r tools/optional-requires
```

Seznam zdrojových kódů 3.6: Ryu instalace a spuštění



Obrázek 3.11: Architektura kontroléru Ryu [72]

3.12.3 Virtuální topologie

Nástrojů pro vytvoření virtuální topologie existuje celá řada. Velké množství jich je dostupných jako open-source. V tabulce 3.10 je uveden přehled vybraných open-source nástrojů pro použití s SDN. Oproti open-source nástrojům existují i komerční virtualizační platformy. Nástroj Cisco Packet Tracer je zdarma dostupný členům Cisco akademie, je více popsán v kapitole 3.12.3.

Virtual topology	Commits	Releases	Forks	Language	Last activity
GNS3	4819	144	288	Python	2Q 2020
Mininet	1965	13	1300	Python	2Q 2020
MidoNet	12421	122	90	Scala	4Q 2019
vSDNemul	118	2	2	-	4Q 2019
Packet Tracer	Komerční nástroj, poslední verze 7.3.0				

Tabulka 3.10: Přehled vývoje nástrojů pro virtualizaci sítě (aktualizace k 9.5.2020)

GNS3

GNS3 je nástroj používaný k emulaci, konfiguraci a testování virtuálních sítí propojených s fyzickou topologií. Virtualizační nástroj umožňuje provozovat menší topologii skládající se pouze z pár zařízeních. Tato simulovaná zařízení je možné připojit na HW síťová zařízení. Virtuální zařízení jsou hostovaná na serverech nebo v cloudu. GNS3 je open-source nástroj. Je aktivně vyvíjen a podporován a má rostoucí komunitu s více než 800 000 členy.

Architektura GNS3 se skládá z grafického rozhraní (GUI) a virtuálního prostředí složeného z virtuálních instancí síťových zařízeních. Jedná se o klientskou část GNS3, pomocí které je vytvořena požadovaná topologie.

Serverovou část je možné realizovat několika způsoby, kterými jsou *Local GNS3 server, Local GNS3 VM a Remote GNS3 VM. Local GNS3 server* běží na stejném hostiteli jako klientská aplikace GNS3. Při použití *Local GNS3 VM* je možné spustit server přímo na lokálním počítači pomocí virtualizačního softwaru, kterým může být například VMware Workstation nebo Virtualbox. Poslední možností je *Remote GNS3 VM*, která se spouští vzdáleně na serveru pomocí VMware ESXi nebo z cloudu [28].

```
1) Ubuntu
sudo add-apt-repository ppa:gns3/ppa
sudo apt update
sudo apt install gns3-gui gns3-server

2) Debian
# pridani zaznamu do~/etc/apt/sources.list
deb http://ppa.launchpad.net/gns3/ppa/ubuntu trusty main
deb-src http://ppa.launchpad.net/gns3/ppa/ubuntu trusty main

# instalace GNS3
sudo apt-get update
sudo apt-get install -y gns3-gui gns3-server

3) Pypi
pip3 install gns3-server
pip3 install gns3-gui
```

Seznam zdrojových kódů 3.7: GNS3 instalace a spuštění

Mininet

Mininet je emulátor sítě, který v sobě současně implementuje i orchestraci emulované sítě. Při spuštění Mininetu je současně spuštěno několik síťových zařízeních, kterými mohou být hosté, přepínače, virtuální propoje a kontroléry. Celý nástroj běží na základě virtualizace v jádře operačního systému. Host v síti Mininet se chová stejně jako reálné zařízení. Na tyto stroje je možné připojit se pomocí SSH a současně je možné spustit na těchto hostech i jednoduché aplikace [46].

Několik výhod, proč je Mininet pozitivně hodnocen širokou komunitou:

- Rychlost - Zapnutí jednoduché sítě je možné v několika sekundách, během kterých se spustí celá topologie, propojí se zařízení a otestuje se funkčnost komunikace.

- Vlastní topologie - Poskytování možnosti tvorby vlastní topologie dává neomezené možnosti v počtu prvků nebo v jejich konfiguraci. Pomocí této vlastnosti je možné simulovat jakoukoliv síť pro testování aplikací.
- Úprava odesílání paketů - Přepínače v Mininetu je možné programovat pomocí OpenFlow protokolu, a tím upravovat komunikaci a tok dat mezi porty.
- Spuštění Mininetu na osobním počítači - Nástroj Mininet je možné spustit ve VM na kterémkoliv počítači případně na serveru, pokud je potřeba vyššího výkonu.
- Open-source - Vývoj tohoto projektu má velmi aktivní komunitu, která vyvíjí nové verze Mininetu. S tímto přístupem je možné si virtuální topologii upravovat například pomocí skriptů v Pythonu.

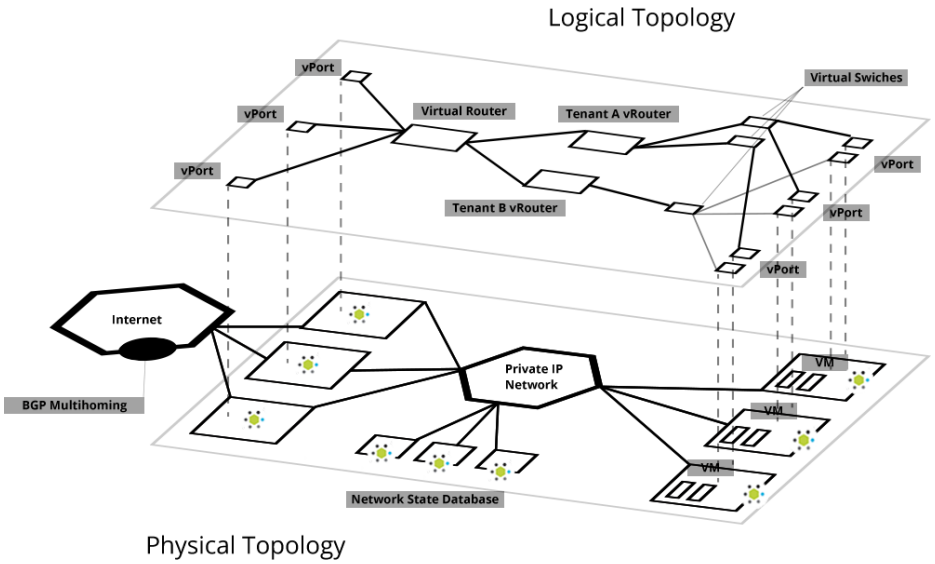
```
# 1) instalace Mininetu ze zdrojoveho kodu
git clone git://github.com/mininet/mininet
cd mininet

# zobrazeni dostupnych verzi
git tag

# Mininet_ver je verze, kterou chcete nainstalovat
git checkout -b Mininet_ver Mininet_ver
cd ..

# instalace Mininetu
mininet/util/install.sh [options]

# options:
# -a: instalace vsech dostupnych soucasti (Mininet, Open vSwitch, OpenFlow Wireshart, POX)
# -nfv: instalace Mininet, OpenFlow reference switch, Open vSwitch

# 2) instalace z balicku (Ubuntu)
sudo apt-get install mininet

# spusteni a otestovani funkcnosti
sudo mn --test pingall
```

Seznam zdrojových kódů 3.8: Mininet instalace a spuštění

MidoNet

MidoNet je open-source platforma pro virtualizaci síťové infrastruktury pro IaaS cloudový software, jako je OpenStack. MidoNet umožňuje operátorům vytvářet izolované sítě v softwaru, které překrývají hardwarovou síťovou infrastrukturu. V OpenStacku nahrazuje výchozí plugin Open vSwitch pluginem MidoNet. Tento produkt je navržen pro distribuované výpočty a zajišťuje síťovou kontrolu sítě podle požadavků, čímž je možné vytvářet a měnit síťovou topologii za běhu [44].

Nejzajímavějším vedlejším účinkem použití škálovatelného distribuovaného systému jsou ekonomické úspory elektřiny. Místo odesílání veškerého provozu prostřednictvím síťových



Obrázek 3.12: Promítnutí logické topologie MidoNet na fyzickou topologii [43]

proprietárních síťových zařízeních se konfigurace a nastavení provozu virtuálních strojů zpracovává v hypervisoru, kde je umístěn VM. Provoz je následně posílán přes servery, které pracují jako L3 brány a *loadbalancery* zátěže. Díky tomuto způsobu řízení provozu operátoři škálují provoz tak, aby respektovali životní prostředí a neplýtvali elektrickou energií. MidoNet podporuje technologii VXLAN (Virtual Extensible LAN) pro vytvoření virtuálních tunelů napříč logickou topologií mezi jednotlivými hosty. Technologie VXLAN umožňuje škálovat až 16 mil. logických sítí [43].

vSDNemul

vSDNemul je nástroj pro vývoj SDN technologií založený na virtualizační platformě Docker. Tento nástroj vytváří virtuální počítače, přepínače, směrovače nebo jiná síťová zařízení pro vytvoření vlastní topologie [23].

```
# pozadavky pro beh aplikace
- Docker (17.12.0-ce or later)
- Python (3.6 or later)
- OpenvSwitch (2.9 or later)
- IPRoute2 (4.11.0 or later)
- Ryu (4.28 or later)
- Any Linux OS (4.3 or later)

# instalace nastroje vSDNemul
git clone https://github.com/fernnf/vsdnemul.git
cd vsdnemu
chmod +x install.sh
./install.sh all

```

```
15 # spusteni a otestovani funkcnosti
16 cd examples
17 sudo python3 simplotopo.py
```

Seznam zdrojových kódů 3.9: vSDNemul instalace a spuštění [23]

Cisco Packet Tracer

Cisco Packet Tracer je výkonný simulační nástroj, který byl vytvořen Cisco Systems pro studenty síťové akademie. Nahrazuje fyzická zařízení v učebně, a tím umožňuje studentům vytvořit ve virtuálním prostředí téměř neomezený počet virtuálních zařízeních. Nástroj umožňuje studentům nacvičit si praktické zkušenosti zahrnující *troubleshooting*.

Packet Tracer je používán ve značném množství kurzů Cisco akademie. Stáhnout si ho a následně používat je možné bez poplatků. Licenčně je dostupný pouze pro nekomerční a neprodukční prostřední určené pouze k procvičování. Packet Tracer může být použit pouze studenty, kteří jsou zařazeni v některém z Cisco kurzů akademie [10].

Prostředí je vytvořeno interaktivně pomocí funkce *drag-and-drop*. Virtuální prostředí současně obsahuje i celou řadu modelů, které je možné využít k vizuální reprezentaci dané části sítě. Mezi modely patří budovy, města, cloudy a další prostředí.

Packet Tracer je dostupný ve dvou režimech. V režimu *real-time* mají všechna zařízení reálnou odezvu, což dává značnou možnost nasimulovat podobně fungující síť stejně, jako na HW zařízeních. Druhým režimem je simulace, ve které se snadno monitorují časové intervaly, propagace dat v síti a další praktické detaily přenosu informací.

Zařízení, se kterými se zde pracuje, jsou modulární. Je tedy možné je virtuálně vypnout a zapnout, umístit do nich rozšiřující moduly, případně jinak upravit.

Pro potřeby akademie a instruktorů je k dispozici *Activity Wizard*, pomocí kterého se snadno vytvoří vlastní výukový plán. Bez problémů se zde vytvoří testovací scénáře a topologie, na které se pracuje v úloze. Tento nástroj současně dovoluje i průběžné hodnocení a zpětnou vazbu při zpracování dílčích úkolů [57].

3.12.4 Zhodnocení výběru vhodných nástrojů pro implementaci

Z přepínačů byl vybrán virtuální přepínač Open vSwitch (viz kapitola 3.12.1), který je plně programovatelný a přizpůsobitelný potřebám vycházejících z modelů a diagramů koncepce pro automatizovaný systém. Open vSwitch disponuje velmi dobře popsanou dokumentací, která je nezbytná pro správné nasazení.

Pro virtuální topologii byl zvolen open-source projekt Mininet (viz kapitola 3.12.3). Mininet je implementován do celé řady testovacích skriptů k open-source SDN kontrolérům, současně obsahuje dostatečnou dokumentaci a širokou komunitu. Je možné dohledat celou řadu způsobů použití a implementací.

Během testování dostupných projektů popsaných v kapitolách výše byly vybrány pro následnou implementaci kontroléry ONOS, OpenDaylight, Ryu, Floodlight a POX (viz kapitola 3.12.2). Tyto kontroléry jsou velmi rozdílné co se týká implementace komponent, způsobu komunikace s nimi a existencí dostatečně podrobné dokumentace.

Poznámky z instalace, testování a výběru vhodných projektů jsou uvedeny v příloze F.

3.13 Vývoj aplikace

3.13.1 Vývojové nástroje

Pro tvorbu samotné aplikace byl vybrán programovací jazyk Python. Volba programovacího jazyka vychází ze zprávy The State of the Octoverse společnosti GitHub Inc. [27], která uvádí přehled projektů vytvořených a udržovaných na platformě GitHub. Jedním z nejvíce používaných jazyků je právě zmíněný jazyk Python. K tomuto jazyku je současně přítomné značné množství knihoven pro práci se síťovou technologií [61].

Většina projektů zaměřená na práci s virtuální topologií (viz tabulka 3.10) je napsána právě v tomto jazyce. Oproti tomu SDN kontroléry ve velké míře používají jazyk Java (viz tabulka 3.4). Python a Java jsou mezi sebou vzájemně kompatibilní. Je tedy možné z Python prostřední spustit Java aplikaci například pomocí Python knihovny *subprocess*. Pro vizuální podobu aplikace je použita knihovna tkinter [62]. Balíček tkinter je standardní rozhraní Pythonu v sadě nástrojů tk GUI. Tk i tkinter jsou k dispozici na většině unixových a Windows platformách.

3.13.2 SDN a virtualizační nástroje

Pro virtuální topologii je použit projekt Mininet [47]. Mininet je open-source projekt určený pro virtualizaci sítě uzpůsobený k vývoji, testování a *demoukázkám* speciálně pro OpenFlow a SDN kompatibilní zařízení. K tomuto projektu je dostupná poměrně podrobně popsaná dokumentace, což je značnou výhodou. Současně je dostupné API pro komunikaci s virtualizovanou síti z aplikace třetí strany. Mininet je použit ve verzi 2.2.2-2.

SDN kontroléry jsou použity ONOS, OpenDaylight, Floodlight, POX, Ryu. Z testování jednotlivých verzí SDN kontrolérů vyšly zmíněné kontroléry jako nejlépe použitelné z ohledu na přítomnost alespoň dobré dokumentace, funkčního API pro komunikaci s nimi a podpory Python případně Javy pro jejich spuštění. Kontroléry byly vybrány pro následující použití v těchto verzích:

- ONOS 1.15.0
- OpenDaylight Carbon
- Floodlight 1.2
- POX eel
- Ryu 4.34

Implementace rozhraní pro REST volání k uvedeným SDN kontrolérům je uvedena v příloze G.

Celé řešení je distribuováno pomocí virtuálního stroje z důvodu přenositelnosti, možnosti zálohování a tvorby *snapshotů*. K práci s virtuálním strojem byl použit nástroj Oracle VM VirtualBox [53], který je dostupný pod licencí GPLv2. K nástroji je udržována přehledná dokumentace.

3.13.3 Softwarová architektura

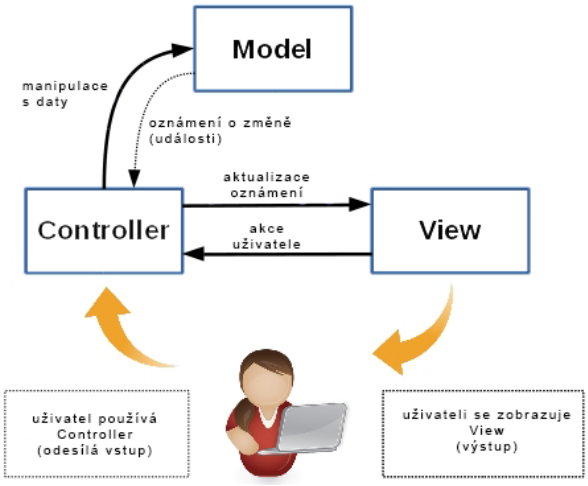
Dle ANSI/IEEE 1471-2000 je softwarová architektura struktura komponent systému, jejich vzájemné vazby, principy a předpisy definující jejich návrh a vývoj v průběhu času [32].

Při zkoumání principů různých druhů návrhových vzorů byla zvolena s ohledem na jednoduchost a vhodnost implementace na základě definovaných vlastností systému (viz kapitola 3), architektura MVC (Model, View, Controller). Tato architektura se skládá ze tří částí, kterými jsou model, uživatelské rozhraní a kontrolér.

Komponenta *model* pracuje s business logikou a nemá žádné informace o komponentě *view* nebo *controller*. Když se data v modelu změní, informuje pomocí *listenerů* o změně.

Komponenta *view* je uživatelské rozhraní. Je odpovědná za zobrazování aktuálního stavu modelu uživateli. Při práci uživatele s grafickým zobrazením se aktualizuje podle dané události.

Poslední komponentou tohoto modelu je *controller*, který se stará o propojení obou výše zmíněných komponent. Monitoruje aktivitu uživatele v grafickém zobrazení a na pozadí zpracovává události vzniklé aktivitou uživatele. Při změně modelu kontrolér aktualizuje uživatelské rozhraní. Na obrázku 3.13 je schematicky zobrazena komunikace mezi těmito třemi komponentami [49].



Obrázek 3.13: MVC architektura [49]

3.13.4 Proces vývoje software

Pro vývoj a realizaci softwaru existuje celá řada modelů. Pro diplomovou práci bylo vybíráno mezi modely vodopád (*waterfall*), iterativním a agilním přístupem. Vývoj automatizovaného systému pro výuku je dynamický a mění se dle možností, které jsou nabízeny. Z tohoto důvodu se nejeví jako vhodné použít vodopádový přístup.

Další dva přístupy jsou velmi podobné. Pro vývoj byly stanoveny kratší úseky s jasně definovanými iteracemi, které je nutné dodržet do daného termínu. Krátké úseky se nazývají *sprinty*. Změny mezi jednotlivými funkcionalitami není nutné zveřejňovat při další verzi,

jako tomu je u iterativního přístupu. Současně plánování dalšího postupu probíhá v kratším časovém úseku, a proto je možné velmi dynamicky systém upravovat, například během dalšího *sprintu*. Nevýhodou agilního přístupu je například problém s možností zanesení chyby do kódu při další iteraci nebo s průběžným kvalitním udržováním dokumentace.

Agilní přístup se jeví jako vhodný k použití na produktový vývoj a vývoj menších projektů. Menšími projekty mohou být části procesů, které jsou následně propojeny do funkčního systému.

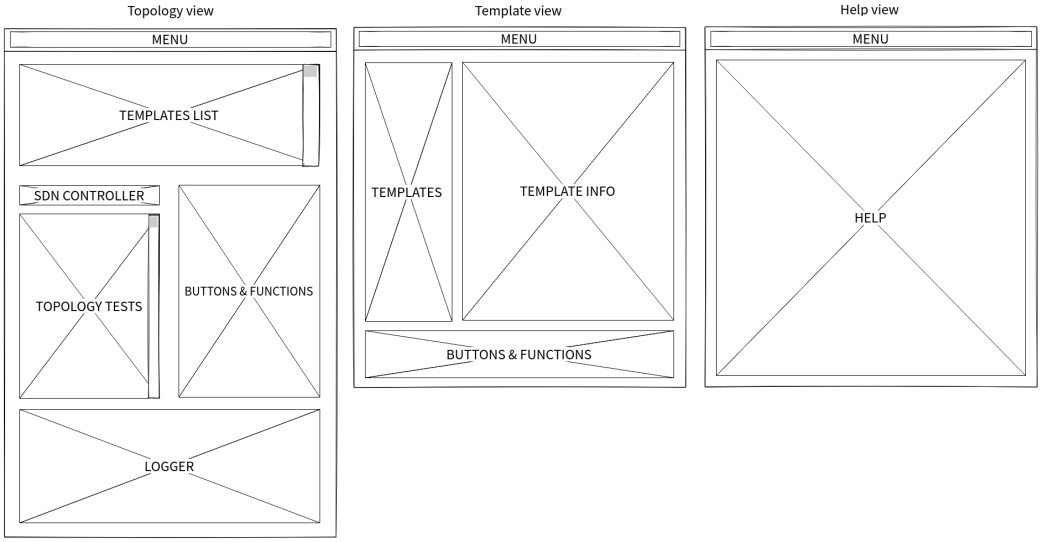
Na základě výše popsaných výhod a nevýhod byl pro vývoj automatizovaného systému zvolen agilní přístup vývoje software.

3.14 Návrh vzhledu systému

Pro návrh vzhledu aplikace byl použit drátový model (*wireframe*). Tímto pojmem rozumíme jednoduchý model, který definuje obsah, rozložení a funkce aplikace. Drátový model je architektonický návrh, který nepotřebuje barvy, obrázky, ani jiný konkrétní obsah. Slouží pro ujasnění rozvržení a funkcí elementů v aplikaci zejména mezi programátorem a zákazníkem.

Během postupného vývoje tvorby aplikací byly vytvořeny čtyři druhy drátových modelů. Jsou jimi textový *wireframe*, blokový *wireframe*, podrobný *wireframe* a interaktivní *wireframe*. Pro tuto diplomovou práci byl zvolen blokový *wireframe*, který dostatečně popisuje funkcionalitu aplikace. Pro tvorbu drátového modelu byla použita online aplikace MockFlow [2].

Systém je z pohledu *frontendu* rozdělen do několika funkčních oken, která se váží k funkcionalitě samotného systému. Okna jsou celkem tři a jsou rozdělena podle vlastní funkcionality, (viz obrázek 3.14). Základními dvěma okny jsou *Topology* a *Template*. Posledním oknem je nápověda, ve které je navržený prostor pro informace o používání systému.



Obrázek 3.14: Drátový model GUI systému

3.14.1 Topologie

Na obrázku 3.14 je znázorněno okno *Topology*, které se otevře po spuštění aplikace. V tomto okně je možné spustit virtuální topologii, která je definována šablonou. Spustit je možné nejen celou topologii, která obsahuje virtuální síť a zvolený SDN kontrolér, ale také jednotlivé komponenty samostatně. Současně je zde možné zvolit SDN kontrolér ze seznamu dostupných modulů v aplikaci. Jelikož je systém navržen jako modulární, je umožněno přidávat další moduly pro ovládání dalších SDN kontrolérů. Ve spodní části se nachází textové okno pro výpis aktuálního stavu virtuální topologie.

3.14.2 Šablony topologie

Pro spuštění topologie je nutné nejprve vytvořit šablonu topologie, což je možné pomocí okna pro správu šablon zobrazeného na obrázku 3.14. Toto okno je pojmenované *Template*. Šablonu je možné vytvořit novou, editovat stávající, případně vymazat některou z již vytvořených.

3.14.3 Nápověda

V okně *Help* (viz obrázek 3.14) jsou uvedeny veškeré potřebné informace o programu a jeho ovládání. Je zde přítomen seznam parametrů, které je možné uvádět do šablon. Součástí nápovědy je i odkaz na postup pro přidání nového kontroléru, případně jak používat rozhraní pro komunikaci se stávajícím kontrolérem. Na konci textu jsou uvedeny licenční podmínky a odkaz na repozitář.

Součástí nápovědy jsou i odkazy na dokumentace jednotlivých projektů, které jsou tímto automatizovaným systémem využívány.

3.15 Ovládání aplikace

Ovládání systému SDN Automation System probíhá pomocí grafického uživatelského rozhraní, které je vytvořené v grafické Python knihovně tkinter. Aplikace je rozdělena do dvou hlavních oken, kterými jsou správa topologie (viz kapitola 3.15.1) a správa šablon topologie (viz kapitola 3.15.2). Grafická podoba systému je umístěna v příloze C.

3.15.1 Správa topologie

Hlavní okno, které se zobrazí jako výchozí po spuštění systému, je okno pro správu topologie. V tomto okně je možné vybrat parametry pro spuštění topologie, spustit celou topologii nebo jen její části. Po spuštění celé topologie je postupně spuštěn vybraný SDN kontrolér a virtuální síť. Po spuštění těchto prvků topologie je načten seznam dostupných testů pro vybranou šablonu topologie.

Funkcionality správy topologie

V níže uvedeném seznamu jsou vypsány veškeré funkcionality, které jsou dostupné v tomto okně systému.

- **Výběr šablony topologie** - V seznamu v horní části okna je nutné před samotným spuštěním virtuální sítě nebo SDN kontroléru vybrat šablonu topologie. V této šabloně jsou uvedeny konfigurační parametry pro komponenty spouštěné v následujících krocích. Šablony jsou definované ve formátu YAML a jsou uloženy v adresáři *topology_templates_config*. Šablony jsou pojmenovány podle praktických zadání.
- **Výběr SDN kontroléru** - V rozbalovacím menu je následně nutné vybrat SDN kontrolér, který se bude spouštět pro vybrané zadání. Pro většinu šablon je možné spustit jakýkoliv SDN kontrolér. SDN kontroléry mají definované nastavení v každé šabloně topologie, ve které je s kontrolérem možné pracovat.
- **Xterm** - Před samotným spuštěním prvků topologie je možné označit volbu *Run hosts with XTerm*, která podle názvu naznačuje, že po spuštění virtuální sítě bude otevřeno konzolové okno pro každé virtuální zařízení, které je definované ve virtuální síti, na kterou odkazuje šablona topologie.
- **Spuštění topologie** - Spustit celou topologii obsahující SDN kontrolér a virtuální síť je možné pomocí stisknutí tlačítka *Run Topology*. Aby bylo možné tuto funkci provést, je nutné vybrat šablonu topologie a SDN kontrolér. Prvky topologie jsou spouštěny v pořadí SDN kontrolér a následně virtuální síť. Mezi těmito dvěma komponentami je definovaná prodleva k ustálení stavu topologie.
- **Spuštění SDN kontroléru** - Mimo spuštění celé topologie je možné spustit pouze SDN kontrolér. Jelikož jsou parametry pro jeho spuštění definované v šabloně topologie, je taktéž nutné vybrat šablonu topologie a SDN kontrolér. Spustit pouze SDN kontrolér je možné pomocí stisknutí tlačítka *Run SDN Controller*. Spuštění samotného SDN kontroléru je vhodné pro ta zadání, která mají definované úkoly pouze pro nastavení SDN kontroléru a nikoliv pro virtuální síť.
- **Spuštění virtuální sítě** - Stejně tak, jako lze spustit pouze SDN kontrolér, je také možné spustit pouze virtuální síť. Pro spuštění virtuální sítě je nutné vybrat šablonu, kde je definována její konfigurace. Spuštění virtuální sítě je možné stisknutím tlačítka *Run Virtual Network*. Virtuální síť může být funkční i bez spuštěného SDN kontroléru. Zadání, která nepracují s SDN kontrolérem, definují v šabloně topologie pouze testy pro celou virtuální síť.
- **Vypnutí topologie** - Tato funkcionalita zajišťuje vypnutí všech procesů, které mohou být zapnuté. Při stisknutí tlačítka *End Topology* je vyvolána funkce, které vypíná veškeré běžící instance všech implementovaných kontrolérů, virtuální síť a současně spouští funkci pro odstranění a navrácení konfigurací, které mohly být nastaveny pomocí nástroje Mininet případně Open vSwitch.
- **Otevření GUI SDN kontroléru** - Pokud je dostupné grafické rozhraní u spuštěného kontroléru, je toto rozhraní otevřeno. Většina SDN kontrolérů má své grafické rozhraní

dostupné pomocí prohlížeče webových stránek, je tudíž otevřen prohlížeč s konkrétní URL adresou. GUI kontroléru se otevře stisknutím tlačítka *Open SDN Controller GUI*.

- **Otestování topologie** - Další významnou funkcionalitou je otestování celé topologie. Testy topologie jsou definovány v samotné šabloně. Podle spuštěného SDN kontroléru a vybrané šablony jsou naimportovány uživatelské testy do seznamu testů v levé části okna. Po stisknutí tlačítka *Test Topology* je provedeno testování, ve kterém jsou postupně spouštěny všechny testy. Podle úspěšnosti daného testu je v seznamu testů daný test označen „OK“, případně „—“. Mezi testy je nastavena časová prodleva, která zajistí konvergenci sítě před spuštěním dalšího testu. Testování sítě je možné spustit v jakémkoliv stavu topologie, před konfigurací, během konfigurace a po dokončení konfigurace dle zadání. Přechody mezi stavy znázorňuje stavový diagram pro práci s topologií (viz obrázek 3.1).

- **Spuštění vlastního skriptu** - Pro splnění definovaného zadání pomocí Python skriptu je připravena funkcionalita systému, která provádí automatizované spuštění vlastního skriptu. Spustitelné vlastní skripty mají definovaný adresář `exercises`. Pro spuštění skriptu je nutné stistknout tlačítko *Run script*. Následně bude spuštěn skript se jménem definovaném v textovém poli nad tímto tlačítkem.

3.15.2 Správa šablon topologie

V okně pro správu šablon topologie je možné prohlížet vytvořené šablony topologie, editovat je a případně přidat novou šablonu založenou na výchozí šabloně. Okno pro správu šablon topologie je možné otevřít kliknutím na tlačítko *Templates* v menu v horní části GUI.

Funkcionality správy šablon topologie

V následujícím seznamu jsou uvedeny funkcionality dostupné v okně správy šablon topologie.

- **Text šablony topologie** - V seznamu šablon topologie v levé části okna jsou vypsány a alfabeticky seřazeny všechny dostupné šablony topologie umístěné v adresáři */topology_ templates_ config/*. Všechny šablony jsou ve formátu yaml, aby byla možná jednoduchá konverze mezi formáty yaml-json.

- **Úprava šablony topologie** - Po výběru šablony topologie a kliknutím na tlačítko *Edit template* se nahraje obsah šablony do obou textových polí umístěných na pravé straně okna. V horním okně jsou přehledné informace o šabloně, jako je popis šablony, její autor, verze šablony, případně verze OpenFlow protokolu. Ve spodním textovém okně jsou veškeré dostupné informace o šabloně. V tomto okně je rovněž možné šablonu editovat. Poslední textovou částí, která se po načtení šablony aktualizuje, je jméno šablony umístěné v horní části okna. Jméno šablony odpovídá názvu souboru v adresáři. Po dokončení úprav v šabloně a stisknutím tlačítka *Save template* je šablona v adresáři aktualizována pod stejným jménem, pod kterým byla původně uložena.

- **Vytvoření nové šablony** - Pro vytvoření nové šablony je nutné stisknout tlačítko *Load default template*. V tuto chvíli se do textových polí v pravé části obrazovky nahrají informace definované ve výchozí šabloně topologie. Každá šablona dodržuje danou strukturu, podle které systém následně načítá jednotlivé komponenty, se kterými pracuje. Jméno šablony je automaticky vygenerováno v podobě *template_ 'timestamp'*, kde časové razítko je ve formátu *%Y%m%d-%H%M%S*. Jméno šablony je možné změnit úpravou textu v poli označeném jako *Template name*. Pod tímto názvem je šablona následně uložena v adresáři s příponou souboru yaml. Nově vytvořená šablona je automaticky přidána do seznamu šablon v levé části okna.

  Pro správné vytvoření topologie je nutné dodržet způsob zápisu pro soubory definované formátem yaml [54]. Pokud je během vytváření šablony zanesena chyba, při dalším načtení šablony k editování je uživatel informován o problému s načtením této šablony.

- **Smazání šablony** - Smazání šablony probíhá výběrem požadované šablony v seznamu šablon a následným kliknutím na tlačítko *Delete template*. Po kliknutí na toto tlačítko je uživatel dotázán, zda opravdu chce smazat vybranou šablonu pomocí dotazovacího dialogu ANO/NE (viz obrázek 3.15). Po stisknutí tlačítka ANO je šablona trvale smazána z adresáře a z celého projektu. Seznam šablon je následně aktualizován. Při stisknutí tlačítka NE, je proces mazání šablony ukončen a vybraná šablona zůstane v nezměněné podobě.

Obrázek 3.15: Dotazovací okno pro smazání šablony topologie

Kapitola 4

Zadání praktických cvičení

Součástí této kapitoly jsou vytvořená zadání pro výuku softwarově definovaných sítí. Jednotlivá zadání jsou koncipována tak, aby studenta provedla problematikou SDN a student se sám dokázal zamyslet nad řešeným problémem. Součástí zadání jsou otázky reagující na konfigurovanou funkcionalitu.

4.1 Způsob výuky

Automatizovaný systém pro výuku SDN technologií je zaměřen primárně na praktickou část problematiky. V jednotlivých zadáních je na začátku kapitoly uveden krátký úvod do tématu, který studentovi objasní řešenou konfiguraci. Zadání jsou vytvořena formou úkolů, které na sebe logicky a systematicky navazují. Každé zadání je na konci doplněno o dotazy na provedenou konfiguraci. Dotazy reagují na kritické konfigurační sekce, se kterými se student setkal v průběhu práce. Pokud je během konfigurace objeven problém, je vhodné postupovat podle *best practice*, případně dohledat k dané problematice vhodný *workaround*.

4.2 Definice praktických zadání

Každé zadání reaguje na jinou problematiku z oblasti softwarově definovaných sítí. Student se v průběhu práce seznámí se způsobem konfigurace SDN kontrolérů a virtuálních sítí, které obsahují koncová zařízení, přepínače a propoje mezi nimi navzájem. Každé ze zadání je spojené s konkrétní topologií. Před spuštěním topologie je nutné vybrat SDN kontrolér, se kterým bude virtuální síť komunikovat. Pro každé zadání je volitelné, který SDN kontrolér bude spuštěn. Ne všechny kontroléry jsou však vhodné pro všechna zadání. Je to dáno například různými verzemi OpenFlow protokolu, které SDN kontroléry podporují.

Všemi implementovanými kontroléry je podporován protokol OpenFlow, byl tudíž použit pro vytvoření komunikačního rozhraní mezi kontrolérem a zařízeními v síti. V každé šabloně topologie je definována sada příkazů a balíčků komponent, se kterými se spouští daný SDN kontrolér. Současně je zde uvedeno nastavení pro virtuální síť, jako je například verze OpenFlow protokolu. V šabloně se v neposlední řadě nachází i seznam testů, včetně očekávaných návratových hodnot pro splnění správné konfigurace testované funkcionality (viz zdrojový kód 4.1).

```
### TOPOLOGY DESCRIPTION ###
# using Open Flow protocol version 1.3
topologyDescription: "Open vSwitch secure mode"
topologyAuthor: "John Doe"
topologyVersion: "v2.0"
topolofyOFVersion: "OpenFlow 1.3"

### TOPOLOGY SETUP ###
topologyTests:
  - ping 1 10.0.0.1 10.0.0.3 True
  - ping 1 10.0.0.2 10.0.0.4 True
  - flow_rule switch_id 00:00:00:00:00:00:00:01 name flow_mod_1 priority
    32768 True
  - flow_rule switch_id 00:00:00:00:00:00:00:01 name flow_mod_1 priority
    32768 in_port 1 actions output=flood True

### VIRTUAL NETWORK SETUP ###
networkTemplate: network2 # network templates in network_template folder

networkSetup:
  ofVersion: 13 # openflow version
  ipBase: 10.0.0.0/8 # base IP address for hosts
  rootIp: 10.0.0.200/32 # IP for host root
  routesRoot: 10.0.0.0/8
  cleanUp: True # if build now, cleanup before creating?
  inNamespace: False # spawn switches and controller in net namespaces?
  sdnControllerIp: 127.0.0.1 # SDN controller IP
  sdnControllerPort: 6653 # listen port that will be SDN Controller
    listening for OpenFlow protocol 6653, 6654
  autoSetMacs: False # set MAC addrs automatically like IP addresses?
  autoStaticArp: False # set all-pairs static MAC addrs?
  switchType: OVSSwitch
  # OVSBridge - OVSBridge is an OVSSwitch in standalone/bridge mode
  # IVSSwitch - Indigo Virtual Switch
  # OVSSwitch - Open vSwitch switch. Depends on ovs-vsctl, OVSSwitch =
    OVSKernelSwitch

### SDN CONTROLLERS SETUP  ###
Floodlight: "fl_config2"

Onos: "onos_config2"

Opendaylight: "odl_config2"

Ryu:
  - --verbose
  - --observe-links
  - --observe-links
  - ryu/app/gui_topology/gui_topology.py
  - ryu/app/simple_switch_13.py

### SDN CONTROLLERS POST CONFIGURATION ###
sdnControllersPostConfig: "topology3PostConfig"
```

Seznam zdrojových kódů 4.1: Ukázka konfigurační šablony pro topologii zadání

Součástí každé šablony topologie je reference na virtuální sít. Virtuální sítě obsahují koncová zařízení, přepínače, propoje mezi nimi a mohou obsahovat i referenci na SDN kontrolér. K definici virtuální sítě je využito API rozhraní 3. úrovně (viz zdrojový kód 4.2) open-source projektu Mininet, který je popsán v kapitole 3.12.3.

```
from mininet.topo import Topo


class Network3(Topo):

    def build(self):

        # hosts
        host1 = self.addHost('h1')
        host2 = self.addHost('h2')
        host3 = self.addHost('h3')

        # switches
        switch1 = self.addSwitch('s1')

        # links
        self.addLink(host1, switch1)
        self.addLink(host2, switch1)
        self.addLink(host3, switch1)


topos = {'network3': (lambda: Network3())}
```

Seznam zdrojových kódů 4.2: Ukázka definice virtuální sítě za použití mid-level API projektu Mininet

4.3 Struktura zadání

Všechna zadání jsou strukturována stejným způsobem, aby bylo na první pohled zřejmé, co má student konfigurovat a jak má postupovat. Kapitoly v zadání jsou navrženy následujícím způsobem:

- Úvod
- Motivace
- Topologie
- Tabulka adresací
- Úkoly
- Testování topologie
- Zhodnocení

- Doplňující dotazy
- Řešení úkolů pro vybraný kontrolér
- Reference

Každé zadání studenta provede danou problematikou, kterou by měl na konci zadání chápat a být následně schopen samostatně zodpovědět dotazy na konfiguraci a funkčnost topologie. Každé zadání je doplněné sadou testů, které kontrolují studentovu úspěšnost při postupování v konfiguraci virtuální sítě a kontroléru.

4.4 Postup pro vypracování zadání

Pro úspěšné vypracování zadání je vhodné, aby student postupoval dle doporučených kroků vyplývajících ze zadání. Na konci každého zadání je vložen seznam příkazů pro správné splnění všech dílčích úkolů.

Virtuální topologii a SDN kontrolér je možné konfigurovat několika způsoby. Prvním, nejjednodušším způsobem pro konfiguraci, je použití GUI kontroléru. GUI je často zprostředkováno uživateli pomocí prohlížeče, je napsáno v jazyce HTML (Hypertext Markup Language) s použitím CSS (Cascading Style Sheets), případně PHP (Hypertext Preprocessor) frameworku.

Další možností je využít interaktivní CLI, které je v tomto případě spouštěno automaticky (pokud to lze), jak u SDN kontroléru, tak u virtuální sítě. Před samotným spuštěním topologie je možné zaškrtnout volbu *„Run hosts with XTerm“*, díky které jsou otevřena konzolová okna pro všechna spouštěná virtuální zařízení v síti. Ke každé komponentě v síti je možné přistupovat pomocí samostatného CLI.

Pokud chceme přistupovat ke kontroléru nebo k virtuální síti nezávisle na použití CLI, je možné využít API. Každý kontrolér implementovaný v této práci podporuje různé druhy API, avšak většina z nich má dostupnou knihovnu pro REST API. Pomocí tohoto programového rozhraní je umožněno odesílat příkazy PUT, GET, POST a DELETE pro práci s konfigurací kontroléru. Příkazy pro toto rozhraní je možné odesílat pomocí připravené funkcionality v systému, která vyžaduje za vstup JSON případně YAML soubor s konfigurací.

Dále je možné využít i aplikace třetí strany pro komunikaci s REST API, kterou může být například Postman. Kontroléry a virtuální sítě podporují i jiné druhy programového rozhraní, a to Java API. Jelikož je systém napsán v jazyce Python, není tohoto rozhraní využito.

4.5 Praktická cvičení

V této kapitole jsou popsána připravená zadání. Praktická cvičení včetně jejich popisu postupu konfigurace a konfiguračních příkazů jsou uvedena v příloze této práce (viz příloha B).

4.5.1 Zadání 1 - OpenFlow protokol a jeho zprávy

Cílem zadání 1 je seznámit studenta s principem fungování protokolu OpenFlow a s jeho chováním (viz kapitola 2.4). Pro seznámení s OpenFlow protokolem je vhodné zachytit datový provoz v nakonfigurované topologii tak, aby z něj bylo patrné, které typy zpráv a jakým způsobem používá OpenFlow protokol. Pro odchytání zpráv je vhodné použít program Wireshark případně Tcpdump.

Spuštěná virtuální síť nástrojem Mininet umožňuje provést několik bash příkazů pro generování provozu nebo otestování síťové funkcionality. Vhodnými nástroji pro seznámení s fungováním různých protokolů jsou například nástroje *ping, iperf, wget, curl, netperf nebo netcat*.

Student by měl být schopen na konci tohoto zadání zodpovědět, které typy zpráv protokol OpenFlow používá a jaké je jejich využití.

4.5.2 Zadání 2 - Open vSwitch standalone mode

V zadání 2 se student seznámí se způsobem konfigurace přepínače Open vSwitch (viz kapitola 3.12.1) pomocí bash příkazů v CLI. Pro toto zadání bude nutné spustit virtuální síť bez přítomnosti kontroléru a manuálně nakonfigurovat přepínač pro zprovoznění komunikace zařízeních v síti. Přepínač má v tomto případě aktivní jak řídicí, tak datovou rovinu a rozhoduje o způsobu zpracování datového toku zcela samostatně na základě vložených pravidel administrátorem.

Pro práci s Open vSwitchem existuje několik příkazů a mezi nejdůležitější patří následující:

- ovs-vsctl: konfigurace ovs-vswitchd databáze (ovs-db)
- ovs-ofctl: konfigurace a správa OpenFlow přepínačů
- ovs-dpctl: administrace datových cest
- ovs-appctl: dotazování a kontrola ovs daemonů

4.5.3 Zadání 3 - Open vSwitch secure mode

Druhým módem, ve kterém je možné Open vSwitch provozovat, je *secure mode*. V tomto módu je OVS připojen ke kontroléru, který do něj vkládá OpenFlow záznamy. Pro konfiguraci SDN kontroléru je v tomto příkladě využito přístupu pomocí GUI kontroléru a REST volání. Konfigurace je možná oběma způsoby, avšak prostřednictvím GUI není u většiny kontrolérů možné konfigurovat všechny dostupné parametry pro OpenFlow záznamy.

SDN kontrolér je v tomto případně spuštěn bez podpory *samoučícího módu* označovaného jako *Forwarding*. Díky vypnutí této funkce není zprovozněno automatické vkládání OpenFlow záznamů do tabulek přepínače. Aby byla zajištěna funkcionalita sítě, je nutné nakonfigurovat OpenFlow záznamy manuálně. Jelikož je Open vSwitch v módu secure, jsou tyto záznamy do přepínače vkládány z SDN kontroléru.

4.5.4 Zadání 4 - HTTP server a firewall

V zadání 4 se student seznámí s funkcionalitou komponenty firewall na SDN kontroléru, která filtruje provoz podle definovaných restrikcí, kterými mohou být například zdrojová a cílová IP adresa, případně MAC (Media Access Control) adresa, zdrojový a cílový port, použitý transportní protokol nebo ID přepínače pro implementaci pravidla. Tato pravidla jsou uložena na kontroléru a nejsou distribuována na přepínač. Pokud je tedy vyvolána komunikace v síti a přepínač pro ni nemá ve své databázi záznam, je přeposlán dotaz na kontrolér. Podle splnění definovaných pravidel v komponentě firewall je komunikace na přepínači povolena nebo zakázána.

V tomto zadání je nutné nakonfigurovat firewall komponentu tak, aby bylo možné vyvolat na dvou z přítomných hostů ICMP provoz na http server. Jednomu z hostů následně povolit komunikaci na spuštěný http server, který je provozován na portu 80.

4.5.5 Zadání 5 - Směrování datových toků

Pro zprovoznění komunikace mezi zařízeními ve dvou sítích je nutné, aby byla na síťovém zařízení povolena komunikace mezi různými adresními prostory. Z pohledu OpenFlow záznamu se konfigurace směrovače neliší od konfigurace přepínače. Pro využití plné funkcionality SDN kontroléru je možné nastavit ACL (Access Control List), ve kterém jsou zanesena pravidla, která umožňují komunikaci z konkrétních adresních prostorů do jiných adresních prostorů. ACL je stejně jako firewall komponenta SDN kontroléru umožňující filtrování toku dat v síti.

Pro správnou komunikaci je třeba správně nakonfigurovat na hostech směrovací tabulky tak, aby komunikace směřující do jiného adresního prostoru byla správně odeslána.

4.6 Definice vlastního zadání

Systém je připraven na rozšíření a implementaci dalších topologií, definic virtuálních sítí, SDN kontrolérů a testů. Tato funkce systému je jedním z definovaných nefunkčních požadavků (viz kapitola 3.5.1). Systém je připraven na dynamický vývoj potřeb výuky síťových technologií. Po umístění nově vytvořené topologie do připraveného adresáře s ní systém začne automaticky pracovat. Více informací o rozšiřitelnosti systému je uvedeno v příloze E.

Kapitola 5

Testování systému

Kapitola testování systému se věnuje definici pojmů a přehledu testovacích metod. Některé z uvedených metod jsou aplikovány na systém a výsledky jsou následně vyhodnoceny.

5.1 Testování softwaru

Testování softwaru je nezbytnou součástí vývoje jakéhokoliv softwarového produktu. Rizika selhání celého systému nebo jeho částí jsou vysoká a mohou mít negativní dopad na firmu používající daný software v kritické sekci svého působení. Testování systému se provádí na základě specifikací produktu, která vznikají na počátku vývoje. Ve specifikaci produktu jsou uvedeny informace o chování systému, jeho funkcionalitách a vzhledu. Testování tedy slouží jako nástroj k získání informací o kvalitě softwaru, kde nalezené chyby jsou vedlejším produktem této činnosti.

O chybě v softwaru hovoříme tehdy, pokud je splněna jedna nebo více z následujících podmínek [64]:

1. *Software nedělá něco, co by podle specifikace produktu dělat měl.*
2. *Software dělá něco, co by podle údajů specifikace produktu dělat neměl.*
3. *Software dělá něco, o čem se produktová specifikace nezmiňuje.*
4. *Software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.*
5. *Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo jej koncový uživatel nebude považovat za správný.*

Další podstatnou vlastností testování softwaru a jeho výsledků je ustanovení ve skutečnost, že *software dělá co má, a nedělá, co nemá.* Testování lze popsat jako analýzu softwaru s cílem nalézt chyby a problémy. Součástí testování je měření funkcionality a kvality softwaru a zhodnocení atributů a schopností softwaru, zda dosahují požadovaných či akceptovatelných výsledků [4].

5.2 Plán testování

Plán testování by měl být vytvořen na začátku celého procesu testování. Jedná se o stěžejní dokument, který obsahuje harmonogram prací a definici druhů a kategorií testů, které budou provedeny. Plán slouží všem testerům jako podpůrný dokument, ve kterém jsou uvedeny odpovědi na většinu dotazů, které by mohly vzniknout a odkazy na místa v dokumentaci pro testované součásti systému [37].

Plán testování obsahuje základní kategorie, kterými jsou:

- Popis testovaného produktu a odkaz na dokumentaci
- Cíle testování, kterých by mělo být dosaženo, včetně prioritizace testování
- Harmonogram testování s rozvrhem pracovního postupu
- Použité testy z definovaných kategorií testovacích scénářů
- Seznam testovacích případů včetně unikátních identifikátorů v pořadí, v jakém budou testovány
- Definice rizik, které mohou znemožnit testování a návrh, jak případný problém řešit pro minimalizaci škod
- Data nezbytná pro provádění testů
- Požadované výstupy v předem definované struktuře pro další zpracování

5.3 Metody testování

Následující kapitoly definují metody testování v pořadí, v jakém je doporučené provádět jednotlivé testy na vyvíjeném systému. První fází je testování kódu programátorem a poslední fází jsou testy ze strany zákazníka [76].

Testování sestavení

V praxi je tento způsob testování označen jako *„test čtyř očí“*, kde programátoři testují programový kód svého kolegy. Je důležité, aby byl program spustitelný a následná fáze testování mohla plynule navázat. Tato fáze kontroly zdrojového kódu je nejméně finančně nákladná.

Jednotkové testování

Pro objektově orientované jazyky je v této části provedeno testování jednotek, kterými jsou třídy a metody. Testovanou jednotkou se nazývá samostatně testovatelná část zdrojového kódu. Ekonomicky se vyplatí začít provádět unit testování před samotným začátkem vývoje softwaru. Případná chyba nebo změna ve zdrojovém kódu je mnohdy časově a finančně náročnější, než oprava samotného kódu.

Funkční a nefunkční testování

FAT (Factory Acceptance Tests) je sada testů, které jsou prováděny na straně dodavatele, tedy vývojářského týmu. Splnění této fáze je nezbytně nutné pro pokračování další fází, kterou je integrační testování. Ve fázi FAT jsou nejčastěji prováděny funkční a nefunkční testy, které reagují na funkční a nefunkční požadavky na systém.

Integrační testování

V této fázi přichází na řadu testovací tým, který připravuje integrační testy, ve kterých je ověřena bezchybná komunikace mezi jednotlivými komponentami aplikace, aplikací a operačním systémem, případně hardwarem nebo jiným I/O (Input/Output) rozhraním. Integrační testy mohou být jak automatizované, tak manuálně spustitelné. Tuto fázi je množné pro menší projekty vynechat, pakliže budou provedeny testy popsané v dalších kapitolách. Obecně platí, *„čím dříve chybu odhalíme, tím méně to pro nás bude nákladné“*.

Systémové testování

V této fázi je aplikace testována jako jeden funkční celek a současně je poskytnuta výstupní kontrola. Systémové testování je poslední fází testování aplikace před předáním vzniklého softwaru zákazníkovi pro akceptační testování. Provádějí se jak funkční, tak nefunkční testy. Aplikace je testována pro různá chování v různých nasazeních. Pokud je zde objevena chyba, je aplikace předána zpět vývojářskému týmu. Po opravení chyby je stejná funkčnost testována znovu.

Akceptační testování

Akceptační testování probíhá na straně zákazníka. Testy jsou prováděny týmem testerů podle předem připravených testovacích scénářů. Pokud jsou na straně zákazníka objeveny nějaké nesrovnalosti s dokumentací nebo s funkčností systému, je nutné chyby v co nejkratším čase reportovat týmu vývojářů, který danou chybu opraví, a zákazník pokračuje v testování dané funkcionality.

5.4 Implementované testy

Spuštění automatizovaného systému probíhá načtením parametrů v souboru *yaml*. Funkcionality, které jsou uživatelem měněny nejčastěji, jsou z tohoto souboru vyňaty a jejich volba je umístěna do GUI. Možnostmi jsou například volba SDN kontroléru, spuštění celé topologie nebo SDN kontroléru a virtuální sítě samostatně a zobrazení konfiguračních oken CLI pro všechny hosty ve virtuální síti.

5.4.1 Testování sestavení

Assembly testování bylo provedeno několikrát v průběhu vývoje systému. Programový kód a architektura byla konzultována s několika kolegy pro nepřehlédnutí chybných výstupů,

případně neočekávaného chování. Testován byl systém pro sestavení aplikace. Vzniklé problémy byly odstraněny pomocí *debug* režimu.

5.4.2 Systémové testování

Testování systému proběhlo po dokončení implementace všech praktických zadání (viz kapitola 4.5). Systém byl spuštěn podle zadání, byly provedeny všechny definované kroky a následně byla vyhodnocena provedená konfigurace pomocí nástroje pro testování konfigurace implementované v systému. Vzniklé nedostatky byly opraveny. Jako například krátká prodleva mezi spuštěním SDN kontroléru a virtuální sítě pro plné zprovoznění všech aplikací SDN kontroléru.

Během testování byla manuálně testována REST volání pro jednotlivé SDN kontroléry, které jsou využívány v automatizovaném systému. Tato volání byla testována pomocí vytvořených náhodných dat. Náhodná data obsahují záznamy OpenFlow protokolu, které jsou umístěny v OpenFlow tabulkách. Data byla vytvořena podle specifikace verze OpenFlow protokolu.

Testování vytváření šablon topologie proběhlo pomocí využití grafického rozhraní (viz příloha C). Pomocí GUI bylo vytvořeno několik šablon a následně uloženo a využito pro spuštění topologie. Během testování bylo současně vytvořeno i několik šablon manuálně, pomocí vytvoření kopie souboru šablony a jejího umístění do odpovídajícího adresáře s náhodně zvoleným jménem souboru. Při testování šablon systém nevykazoval neobvyklé nebo neočekávané chování.

Součástí testování systému byla vytvořena *appliance* ve formátu `OVA`. Tato *appliance* byla naimportována na jiném zařízení jako virtuální stroj. Funkčnost systému nebyla pozměněna a žádné náhodně vznikající problémy nebyly zaznamenány. Otestování importu virtuálního stroje proběhlo na platformách Windows a Linux v nástroji Oracle VM VirtualBox.

Po dokončení testování v této fázi bylo testování posunuto do fáze akceptačního testování.

5.4.3 Akceptační testování

Pro akceptační testování bylo zvoleno několik studentů z řad vysokých a středních škol technického zaměření. Studenti byli vybráni tak, aby měli znalosti konfigurace počítačových sítích a alespoň základní povědomí o softwarově definovaných sítích. Před samotným testováním byl uživatel seznámen se systémem, jeho účelem a s dokumentací, ve které je uvedeno několik diagramů funkcionalit systému. Součástí představení systému bylo i objasnění práce se šablonami a jejich účel.

Pro akceptační testování jsou připraveny dva scénáře, které uživatele navádí k práci se samotnou topologií a k editaci šablon topologie. Testovací scénář 1 je možné použít pro všechna zadání. Testovací scénář 2 je připravený pro otestování GUI ke konfiguraci šablon.

Testovací scénář 1 - Konfigurace síťových zařízeních v topologii podle zadání

1. Změňte aktuální adresář pomocí zadání příkazu do okna terminálu: `cd /home/user/SDNAutomationSystem/`.
2. Spusťte program SDN Automation System pomocí příkazu: `python3.7 main.py`.
3. V seznamu šablon topologií vyberte šablonu definovanou v zadání.
4. Ve výběru SDN kontroléru vyberte kontrolér dle vlastních preferencí.
5. Stiskněte tlačítko *Run Topology*, které spustí topologii obsahující SDN kontrolér a virtuální síť.
6. Po spuštění všech zařízeních postupujte podle zadání a proveďte potřebnou konfiguraci.
7. Po konfiguraci spusťte otestování topologie pomocí stisknutím tlačítka *Test Topology*.
8. Pokud jsou všechny testy označeny „OK“, zadání je splněno. Vypněte topologii stisknutím tlačítka *End Topology*. Pokud je u některého z testů označení „–“, proveďte *troubleshooting* daného problému.
9. Vypněte SDN Automation System pomocí kliknutí na tlačítko s křížkem v pravém horním rohu GUI.

Testovací scénář 2 - Práce se šablonou topologie

1. Změňte aktuální adresář pomocí zadání příkazu do okna terminálu: `cd /home/user/SDNAutomationSystem/`.
2. Spusťte program SDN Automation System pomocí příkazu: `python3.7 main.py`.
3. Otevřete okno pro správu šablon kliknutím na tlačítko *Template*.
4. Klikněte na tlačítko *Load default template*. Do textových polí se vloží informace z výchozí šablony. Editujte parametry dle potřeby.
5. Po dokončení úprav šablony klikněte na tlačítko *Save*. Šablona se uloží a přidá se do seznamu dostupných šablon v levé části okna.
6. V seznamu existujících šablon zvolte šablonu kliknutím na některý z názvů šablon a následně klikněte na tlačítko *Edit template*. V oknech *Template info* se objeví obsah šablony a informace o ní.
7. Editujte obsah šablony změnou kteréhokoliv parametru. Změňte například verzi OpenFlow protokolu na jednu z hodnot: 10, 11, 12, 13, 14 nebo 15.
8. Po editaci šablony ji uložte stisknutím tlačítka *Save*, šablona se uloží.

9. Znovu v seznamu existujících šablon zvolte šablonu kliknutím na některý z názvů šablon a následně klikněte na tlačítko *Delete template*. V otevřeném dialogu pro potvrzení provedené akce stiskněte tlačítko *ANO* a potvrďte smazání vybrané šablony. Vybraná šablona bude smazána jak ze systému, tak ze seznamu šablon. Zkontrolujte tento stav v seznamu dostupných šablon topologie.

10. Vypněte SDN Automation System pomocí kliknutí na tlačítko s křížkem v pravém horním rohu GUI.

Doplňující dotazy, které jsou testovanému subjektu pokládány v průběhu testování

- Jak byste editovali šablonu pro topologii?
- Kam byste klikli, kdybyste chtěli spustit pouze SDN kontrolér a kam pro spuštění virtuální sítě?
- Co očekáváte, že uvidíte ve spodní části obrazovky v poli pro textový výpis v okně pro správu topologie?
- Jak byste postupovali při ohodnoceném testu „- -“ po stisknutí tlačítka *Test Topology*?
- Kde byste hledali možnosti konfigurace šablony pro topologii v případě vytváření nového zadání?

Na výše uvedené dotazy bylo v souhrnu odpovídáno velmi pozitivně a uživatel byl na první pohled ztotožněn s funkcionalitami systému. V několika případech nebyla odpověď na uvedenou otázku konkrétní, což je zapříčiněno nedostatečnou neznalostí uživatele se spouštěnými SDN kontroléry.

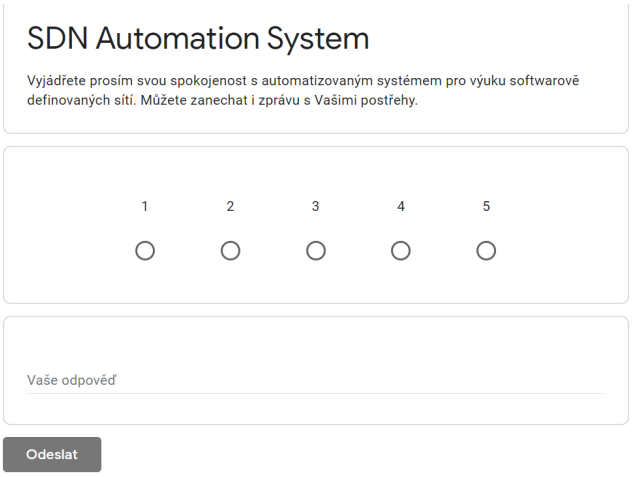
5.4.4 Dotazník hodnocení

Účelem dotazníku hodnocení je zjistit, zda je uživatel spokojený s automatizovaným systémem a s jeho podporou při výuce softwarově definovaných sítí. Dotazník současně uživateli dovoluje vyjádřit svůj názor formou doprovodného textu. Díky tomuto dotazníku je poskytnuta zpětná vazba od uživatelů systému, pro které byl vytvořen. Získané názory pomohou v dalším vývoji softwaru.

Pro dotazník byla stanovena tato stupnice hodnocení:

- 1 - velmi negativní názor
- 2 - negativní názor
- 3 - neutrální názor
- 4 - pozitivní názor
- 5 - velmi pozitivní názor

Cílem je dosáhnout co nejvyššího výsledku hodnocení. Stupnice hodnocení 1-5 je realizovaná pomocí lineárního přepínače s jednou možností odpovědi. Pod stupnicí je dostupné textové pole pro případný vlastní názor. Hodnotící dotazník je zobrazen na obrázku 5.1.



Obrázek 5.1: Hodnotící dotazník

Následující tabulka 5.1 shrnuje vyjádřené odpovědi v dotazníku, které uživatelé odeslali v době testování systému. Tabulka obsahuje vypočítané hodnoty průměru, mediánu a průměrné odchylky z hodnot v tabulce.

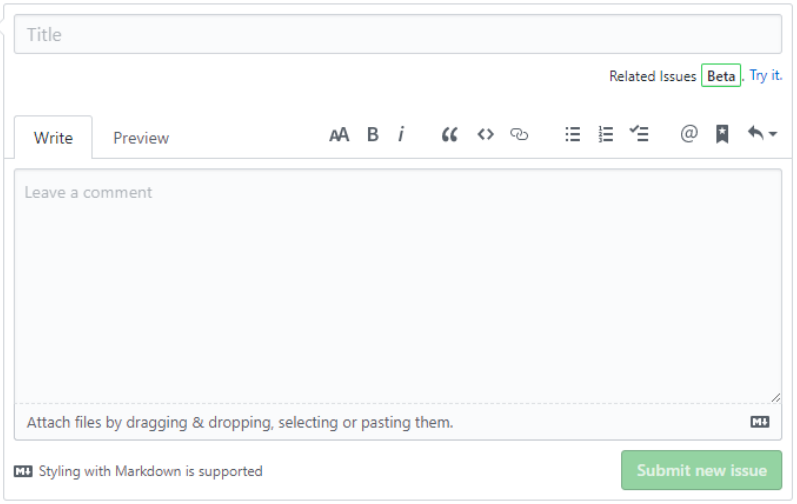
Uživatel	**Hodnocení**
1	4
2	3
3	4
4	5
5	2
6	3
7	3
8	3
9	5
10	3
11	4
12	5
Průměr	3,67
Medián	3,5
Průměrná odchylka	0,83

Tabulka 5.1: Výsledky hodnocení z dotazníku

Z průměru podle stupnice uvedené výše můžeme usoudit, že uživatelé hodnotili systém s výsledkem v intervalu (neutrální; pozitivní). Medián se pohybuje ve stejném intervalu, ale průměrná odchylka naznačuje, že se hodnoty v dotazníku značně liší. Do úvahy je také nutné začlenit i komentáře, které byly připsány při plnění dotazníku.

Z komentářů vyplývá, že uživatel, který hodnotil systém menším počtem bodů, uvedl, že se mu myšlenka automatizovaného systému pro výuku líbí, a proto je vhodné systém dále rozvíjet. V dotazníku se objevily i reakce, které přímo nesouvisí s daným automatizovaným systémem, ale s následnou prací se spuštěnými kontroléry a virtuální sítí. Tyto komentáře naznačují, že uživatel může být při odpovídání ovlivněn fungováním nejen automatizovaného systému, ale také využitými projekty třetí strany.

Další možností, jak zanechat zpětnou vazbu, je nahlášení problému pomocí nástroje *Issues* na platformě GitHub (viz obrázek 5.2). Jelikož je systém sdílen ve veřejném repozitáři, je možné kýmkoliv registrovaným nahlásit problém.



Obrázek 5.2: Nahlášení problému na platformě GitHub

Kapitola 6

Závěr

Hlavním cílem této práce bylo vytvořit softwarový nástroj, který by usnadnil výuku softwarově definovaných sítí, umožňoval automatizované spuštění předem definovaného zadání a poskytl uživateli zpětnou vazbu na provedenou konfiguraci. Tyto požadavky naplňuje vytvořený nástroj `SDN Automation System`. Jedná se o aplikaci napsanou v jazyce Python postavenou na architektuře model-view-controller. Součástí systému je funkcionalita pro automatizované spuštění topologie podle předem definované šablony. Po spuštění systému a provedení konfigurace na síťových zařízeních je uživateli umožněno otestovat nakonfigurovanou topologii (viz kapitola 3).

Pro poskytnutí stejných funkcionalit definovaných v šabloně daného zadání bylo vytvořeno jednotné programové rozhraní pro komunikaci se síťovými prvky. Danou topologii je možné spustit s vybraným SDN kontrolérem. Systém obsahuje třídy umožňující přístup ke konfiguraci SDN kontrolérů pomocí programového rozhraní REST API. Vytvořit univerzální programové vybavení vyvolalo řadu problémů, které se podařilo překonat (viz příloha F).

Před samotnou implementací automatizovaného systému byly analyzovány požadavky na systém (viz kapitola 3.5), které jsou promítnuty do samotné implementace systému. Přehled zařízeních, která jsou kompatibilní s protokolem OpenFlow, je zpracován v kapitole 2. Vybrané a použité komponenty, se kterými systém pracuje, jsou rozebrány následně v kapitole 3.12.

Výsledkem mé diplomové práce je automatizovaný systém pro zavádění praktických zadání v síťových akademiích. Systém minimalizuje čas nutný pro zavedení topologie SDN. Práce obsahuje několik vytvořených zadání, která jsou seřazena podle složitosti v kapitole 4. Pro zavedení nového zadání do systému je nezbytné vytvořit novou šablonu, která implementuje veškerou logiku topologie. Postup vytvoření nového zadání je shrnut v příloze E.

Systém je distribuován jako virtuální stroj, který je snadné nasadit na koncové stanice. Práce je volně dostupná v repozitáři GitHub pod názvem *SDNAutomationSystem* [25]. Repozitář je doplněn o dokumentaci obsahující popis, funkce systému a způsob implementace vlastních šablon v případě rozšíření systému o další funkcionality.

6.1 Přínos práce

Díky nástroji `SDN Automation System` lze výrazně snížit čas nezbytný ke konfiguraci topologie pro přípravu výuky. Topologie obsahuje SDN kontrolér, přepínače, koncová zařízení a propoje těchto zařízení. Současně je minimalizován čas a náklady na lidské zdroje pro následnou kontrolu správnosti řešení daného zadání.

Diplomová práce reaguje na rychle rozvíjející se oblast IT, a to oblast počítačových sítí. Pro udržení dostatečného počtu systémových inženýrů je žádoucí, aby se systémoví inženýři začali seznamovat s konceptem SDN co nejdříve. `SDN Automation System` umožňuje rychle a efektivně nasadit vybranou topologii včetně zpětné vazby na provedenou konfiguraci.

6.2 Budoucí vývoj

- **Rozšíření platformy dostupných SDN kontrolérů** - SDN Automation System je navržený tak, aby umožnil přidání SDN kontrolérů co nejjednodušší formou. Rozšířit systém je možné o kontroléry open-source nebo komerční. Způsob rozšíření systému je popsán v příloze E.
- **Rozšíření sady zadání, virtuálních sítí a testů** - Systém je navržen pro jednoduché přidání nových zadání obsahující virtuální sítě a testy topologie. Pro další vývoj výuky je možné přidat další topologie, na kterých bude probíhat výuka. Způsob rozšíření sady zadání je popsán taktéž v příloze E.
- **Implementace fyzické topologie** - V současné chvíli je implementována virtuální síť z pohledu jednoduchosti spuštění a nasazení virtuálního stroje na klientském počítači. V budoucím rozšíření je možné doplnit systém o komunikaci kontroléru s fyzickou topologií. Další vývoj je možné směrovat i k firmwaru OpenWRT, který podporuje protokol OpenFlow na běžném síťovém zařízení.
- **Oddělení síťových zařízení** - Systém včetně všech použitých zařízeních je implementován ve virtuálním stroji, který je distribuován na koncové stanice. Pro možnosti dalšího rozšíření je možné všechna síťová zařízení oddělit a každé z nich umístit do vlastního virtuálního prostředí.
- **Rozšíření grafického rozhraní** - Další vývoj grafického rozhraní by bylo vhodné zaměřit na zobrazení podrobnějších informací o právě spuštěné topologii, případně o další funkcionality zaměřené na síťová zařízení v topologii. Grafické rozhraní je v současné době vytvořené pomocí knihovny tkinter, tudíž se jedná o aplikaci v okně. Zajímavou myšlenkou by bylo použít pro GUI aplikace webový prohlížeč s přístupem k systémovým zdrojům.
- **Analytické nástroje toku dat** - Jelikož data OpenFlow protokolu obsahují informace pro analytické využití, bylo by vhodné implementovat funkcionalitu pro práci s touto vlastností, analyzovat toky dat a přizpůsobit tomu testy topologií.

Literatura

[1] A. Farrel, Q. Zhao, R. Li, C. Zhou. *An Architecture for Use of PCE and PCEP in a Network with Central Control* [online]. IETF. [cit. 28. 1. 2020]. Dostupné z: <https://tools.ietf.org/id/draft-ietf-teas-pce-central-control-05.html>.

[2] A Produle Systems Pvt Ltd. *MockFlow - The UI planning suite.* [online]. A Produle Systems Pvt Ltd. [cit. 1. 3. 2020]. Dostupné z: <https://mockflow.com/>.

[3] Bc. Viktor Přehnal. *Diagramy nasazení* [online]. Stránky o jazyku UML. [cit. 13. 4. 2020]. Dostupné z: <http://www.jazyk-uml.cz/diagramy-nasazeni/>.

[4] Bohumír Zoubek. *Software testing* [online]. Profinit - new prontier group. [cit. 3. 4. 2020]. Dostupné z: <https://cw.fel.cvut.cz/old/_media/courses/a4m33sep/prednasky/05_testing.pdf>.

[5] CASE, J. et al. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990. Dostupné z: <http://www.ietf.org/rfc/rfc1157.txt>.

[6] Chowdhury, N.M. Mosharaf Kabir; Boutaba, Raouf. *A survey of network virtualization.* Computer Networks, ISSN: 1389-1286, 2010. doi: 10.1016/j.comnet.2009.10.017.

[7] Cisco. *What is SD-WAN?* [online]. Cisco. [cit. 11. 12. 2019]. Dostupné z: <https://www.cisco.com/c/cs_cz/solutions/enterprise-networks/sd-wan/what-is-sd-wan.html>.

[8] Cisco. *Cisco Software-Defined Access, Enabling intent-based networking, 2nd edition* [online]. Cisco. [cit. 17. 12. 2019]. Dostupné z: <https://www.cisco.com/c/dam/en/us/products/se/2018/1/Collateral/nb-06-software-defined-access-ebook-en.pdf>.

[9] Cisco. *Intent-Based Networking (IBN) Explained* [online]. Cisco. [cit. 8. 12. 2019]. Dostupné z: <https://www.cisco.com/c/en/us/solutions/intent-based-networking.html>.

[10] Cisco Networking Academy. *Becoming A Cisco Networking Academy - Frequently Asked Questions* [online]. Cisco Networking Academy. [cit. 30. 1. 2020]. Dostupné z: <https://www.netacad.com/sites/default/files/faq-why-networking-academy.pdf>.

[11] Cisco Systems. *Cisco OpenFlow Agent for Nexus 3000 and 9000 Series Switches* [online]. Cisco Systems. [cit. 1. 3. 2020]. Dostupné z: <https://www.cisco.com/c/en/us/td/d

ocs/switches/datacenter/nexus/openflow/b_openflow_agent_nxos_n3kn9k/b_openflow_native_agent_nxos_7x_chapter_01.html>.

[12] Cisco Systems. *Cisco DNA Center Platform Overview* [online]. Cisco Systems. [cit. 2. 2. 2020]. Dostupné z: <https://developer.cisco.com/docs/dna-center/#!cisco-dna-center-platform-overview>.

[13] Connor Craven. *SD-Branch vs SD-WAN: What's the Difference?* [online]. SDxCentral. [cit. 8. 12. 2019]. Dostupné z: <https://www.sdxcentral.com/networking/sd-wan/definitions/sd-branch-vs-sd-wan/>.

[14] Connor Craven. *What is SD-Branch? Definition* [online]. SDxCentral, LLC. [cit. 14. 12. 2019]. Dostupné z: <https://www.sdxcentral.com/networking/sd-wan/definitions/what-is-sd-branch/>.

[15] CROCKFORD, D. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006. Dostupné z: <http://www.ietf.org/rfc/rfc4627.txt>.

[16] David Čápka. *UML - Use Case Diagram* [online]. itnetwork.cz. [cit. 25. 2. 2020]. Dostupné z: <https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram>.

[17] Department of Computer Science - The McKeown Group. *The basics of SDN and the OpenFlow Network Architecture* [online]. Stanford University. [cit. 31. 3. 2020]. Dostupné z: <http://yuba.stanford.edu/cs244wiki/index.php/Overview>.

[18] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, Steve Uhlig. *Software-Defined Networking: A Comprehensive Survey* [online]. arXiv, Cornell University. [cit. 26. 1. 2020]. Dostupné z: <https://arxiv.org/pdf/1406.0440.pdf>.

[19] Dijiang Huang, Huijun Wu. *Mobile Cloud Computing.* Morgan Kaufmann, 2017. ISBN: 9780128096444.

[20] ENNS, R. et al. Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard), June 2011. Dostupné z: <http://www.ietf.org/rfc/rfc6241.txt>.

[21] Esposito, Flavio; Matta, Ibrahim; Ishakian, Vatche. *Slice Embedding Solutions for Distributed Service Architectures.* ACM Computing Surveys, 2017. doi: 10.1145/2522968.2522974.

[22] FD.io Project. *The Vector Packet Processor (VPP)* [online]. FD.io Project. [cit. 7. 4. 2020]. Dostupné z: <https://fd.io/vppproject/vpptech/>.

[23] Fernando Farias. *vSDNemul* [online]. Github. [cit. 16. 3. 2020]. Dostupné z: <https://github.com/fernnf/vsdnemul>.

[24] flowforwarding.org. *LINC - OpenFlow software switch* [online]. GitHub. [cit. 7. 4. 2020]. Dostupné z: <https://github.com/FlowForwarding/LINC-Switch>.

[25] František Flachs. *Automatizovaný systém konfigurace SDN síťových zařízeních* [online]. GitHub. Dostupné z: <https://github.com/frantisekflachs/SDNAutomationSystem>.

[26] Gephi.org. *The Open Graph Viz Platform* [online]. Gephi.org. [cit. 21. 4. 2020]. Dostupné z: <https://gephi.org/>.

[27] GitHub Inc. *The State of the Octoverse* [online]. GitHub Inc. [cit. 25. 2. 2020]. Dostupné z: <https://octoverse.github.com/>.

[28] GNS3. *Getting Started with GNS3* [online]. GNS3. [cit. 29. 2. 2020]. Dostupné z: <https://docs.gns3.com/>.

[29] GORANSSON, Paul, Chuck BLACK a Timothy CULVER. *Software defined networks: a comprehensive approach, Second edition.* Singapore: Morgan Kaufmann, 2017. ISBN: 0-12804-555-8.

[30] Gunnar Peipman. *Why Azure Rest APIs and How to Prepare for Using Them?* [online]. DZone. [cit. 31. 3. 2020]. Dostupné z: <https://dzone.com/articles/why-azure-rest-apis-and-how-to-prepare-for-using-t>.

[31] Gyewan An. *Netconf, Restconf, grpc-basic* [online]. Mobile Convergence Laboratory. [cit. 31. 3. 2020]. Dostupné z: <https://www.slideshare.net/roy1022/netconf-restconf-grpcbasic>.

[32] IEEE. IEEE Recommended Practice for Architectural Description for Software-Intensive Systems. *IEEE Std 1471-2000.* 2000, s. 1–30.

[33] IEEE. *Recommended Practice for Software Requirements Specifications.* IEEE, 1998. doi: 10.1109/IEEESTD.1998.88286.

[34] IIBY. *Business Analysis Body of Knowledge.* International Institute of Business Analysis, 2015. ISBN: 9781927584026.

[35] Ing. Andrey Shchurov, Ph.D. *Automated Test Design in Multilayer Networks.* Czech technical university in Prague, 2017.

[36] Ing. Andrey Shchurov, Ph.D. *A formal model of distributed systems for test generation missions.* International Journal of Computer Trends and Technology, 2014. DOI: 10.14445/22312803/IJCTT-V15P128.

[37] ISO, IEC, IEEE. *Software and systems engineering — Software testing* [online]. ISO, IEC, IEEE. [cit. 3. 4. 2020]. Dostupné z: <https://www.iso.org/standard/45142.html>.

[38] Ivor Diedricks, Srini Jasti. *Open programmable architecture delivering value beyond connectivity* [online]. Cisco Blogs. [cit. 7. 4. 2020]. Dostupné z: <https://blogs.cisco.com/networking/open-programmable-architecture-delivering-value-beyond-connectivity>.

[39] Lance Boley. *Emulation or virtualization: What's the difference?* [online]. Dell Inc. [cit. 21. 2. 2020]. Dostupné z: <https://blog.dell.com/en-us/emulation-or-virtualization-what-s-the-difference/>.

[40] Liehuang Zhu, Md Monjurul Karim, Kashif Sharif, Fan Li, Xiaojiang Du, Mohsen Guizani. *SDN Controllers: Benchmarking & Performance Evaluation* [online]. arXiv, Cornell University. [cit. 13. 1. 2020]. A VERSION IS UNDER REVIEW AT IEEE JSAC. Dostupné z: <https://arxiv.org/pdf/1902.04491.pdf>.

[41] Luis Sola, Miguel Romance, Regino Criado, Julio Flores, Alejandro Garcia del Amo, and Stefano Boccaletti. *Eigenvector centrality of nodes in multiplex net- works.* Chaos, 23(3):033131, 2013.

[42] Microsoft Corporation. *What is virtualization?* [online]. Microsoft Corporation. [cit. 21. 2. 2020]. Dostupné z: <https://azure.microsoft.com/cs-cz/overview/what-is-virtualization/>.

[43] MidoNet. *MidoNet Home* [online]. MidoNet. [cit. 29. 2. 2020]. Dostupné z: <https://www.midonet.org/#quickstart>.

[44] MidoNet. *MidoNet Wiki* [online]. MidoNet. [cit. 29. 2. 2020]. Dostupné z: <https://github.com/midonet/midonet/wiki>.

[45] Miercom. *Enterprise Campus Architecture Comparative Assessment* [online]. Miercom. [cit. 14. 1. 2020]. Dostupné z: <https://miercom.com/pdf/reports/180830F.pdf>.

[46] Mininet. *Introduction to Mininet* [online]. Mininet. [cit. 29. 2. 2020]. Dostupné z: <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>.

[47] Mininet Team. *Mininet, An Instant Virtual Network on your Laptop (or other PC)* [online]. Mininet Team. [cit. 25. 2. 2020]. Dostupné z: <http://mininet.org/>.

[48] Nefeli Networks, The Regents of the University of California. *BESS (Berkeley Extensible Software Switch)* [online]. GitHub. [cit. 7. 4. 2020]. Dostupné z: <https://github.com/NetSys/bess>.

[49] Ninad Sathaye. *Learning Python Application Development.* Packt Publishing, 2016. ISBN: 9781785889196.

[50] Object Management Group. *Unified Modeling Language* [online]. Object Management Group. [cit. 25. 2. 2020]. Dostupné z: <https://www.omg.org/spec/UML/2.0/>.

[51] OpenDaylight Project a Series of LF Projects. *OpenDaylight Architecture - Operational View* [online]. OpenDaylight Project a Series of LF Projects. [cit. 11. 4. 2020]. Dostupné z: <https://www.opendaylight.org/what-we-do/current-release/sodium>.

[52] OpenWrt Project. *OpenWrt Project* [online]. OpenWrt Project. [cit. 7. 4. 2020]. Dostupné z: <https://openwrt.org/>.

[53] Oracle Corporation. *VirtualBox* [online]. Oracle Corporation. [cit. 25. 2. 2020]. Dostupné z: <https://www.virtualbox.org/>.

[54] Oren Ben-Kiki, Clark Evans, Ingy döt Net. *YAML Ain't Markup Language (YAML) Version 1.2* [online]. yaml.org. [cit. 11. 4. 2020]. Dostupné z: <`https://yaml.org/spec/1.2/spec.html`>.

[55] P4 project - Open Networking Foundation. *BEHAVIORAL MODEL (bmv2)* [online]. GitHub. [cit. 7. 4. 2020]. Dostupné z: <`https://github.com/p4lang/behavioral-model`>.

[56] Paul Goransson, Chuck Black, Timothy Culver. *Software Defined Networks: A Comprehensive Approach.* Elsevier, Morgan Kaufmann, 2014. ISBN: 978-0128045558.

[57] Petr Kudlacek. *Cisco Packet Tracer: Simple Steps to Download the Software For Free* [online]. Software Battle. [cit. 30. 1. 2020]. Dostupné z: <`https://www.softwarebattle.com/cisco-packet-tracer-7-1-1/`>.

[58] Philippa E. Pattison and Stan Wasserman. *Logit models and logistic regressions for social networks: Ii. multivariate relations.* British Journal of Mathematical and Statistical Psychology, 1999.

[59] Podhradský Pavel, Helebrandt Pavel, Halagan Tomáš, Drozd Ivan. *Sítě budoucnosti - SDN a NFV.* České vysoké učení technické v Praze, Fakulta elektrotechnická, 2017. ISBN: 978-80-01-06246-3.

[60] Project Floodlight. *Indigo Virtual Switch* [online]. GitHub. [cit. 7. 4. 2020]. Dostupné z: <`https://github.com/floodlight/ivs`>.

[61] Python Software Foundation. *UsefulModules* [online]. Python Software Foundation. [cit. 25. 2. 2020]. Dostupné z: <`https://wiki.python.org/moin/UsefulModules#Networking`>.

[62] Python Software Foundation. *tkinter — Python interface to Tcl/Tk* [online]. Python Software Foundation. [cit. 25. 2. 2020]. Dostupné z: <`https://docs.python.org/3/library/tkinter.html`>.

[63] Rahamatullah Khondoker, Adel Zaalouk, Ronald Marx, Kpatcha Bayarou. *Feature-based comparison and selection of Software Defined Networking (SDN) controllers* [online]. IEEE. [cit. 14. 1. 2020]. Dostupné z: <`https://ieeexplore.ieee.org/document/6916572`>.

[64] Ron Patton. *Testování softwaru.* Computer Press, 2002. ISBN: 80-7226-636-5.

[65] Rudolph Blair. *Software Defined Networking: OpenFlow Switches & Controllers* [online]. POSTECH, Korea - Department of Computer Science and Engineering. [cit. 7. 4. 2020]. Dostupné z: <`https://slideplayer.com/slide/6204491/`>.

[66] Saleh Asadollahi, Dr. Bhargavi Goswami, Dr. Atul M Gonsai . *Software Defined Network, Controller Comparison.* Queensland University of Technology, 2017. ISSN: 2320-9801.

[67] Sarah Hatton. *Choosing the right prioritisation method.* 19th Australian Conference on Software Engineering, 2008. ISBN: 978-0-7695-3100-7.

[68] SDNCentral LLC. *Special Report: OpenFlow and SDN – State of the Union* [online]. SDNCentral LLC. [cit. 26. 1. 2020]. Dostupné z: <https://www.opennetworking.org/wp-content/uploads/2013/05/Special-Report-OpenFlow-and-SDN-State-of-the-Union-B.pdf>.

[69] SDxCentral. *SDN Controllers Report* [online]. SDxCentral. [cit. 15. 1. 2020]. Dostupné z: <https://www.sdxcentral.com/wp-content/uploads/2015/08/SDxCentral-SDN-Controllers-Report-2015-B2.pdf>.

[70] Snabb. *Snabb: Simple and fast packet networking* [online]. GitHub. [cit. 7. 4. 2020]. Dostupné z: <https://github.com/snabbco/snabb>.

[71] Sreenivas Voruganti, Sriram Subramanian. *Software-Defined Networking (SDN) with OpenStack*. Packt Publishing, 2016. ISBN: 9781786465993.

[72] Sridhar Rao. *SDN Series Part Four: Ryu, a Rich-Featured Open Source SDN Controller Supported by NTT Labs* [online]. The New Stack. [cit. 7. 4. 2020]. Dostupné z: <https://thenewstack.io/sdn-series-part-iv-ryu-a-rich-featured-open-source-sdn-controller-supported-by-ntt-labs/>.

[73] Srini Seetharaman, Anirudh Ramachandran, Sriram Natarajan. *ONOS Tutorial* [online]. SDN Hub. [cit. 11. 4. 2020]. Dostupné z: <http://sdnhub.org/tutorials/onos/>.

[74] Tatu Ylonen. *SSH - Secure Login Connections over the Internet* [online]. SSH Communications Security. [cit. 28. 1. 2020]. Dostupné z: <https://www.ssh.com/ssh/protocol>.

[75] The Linux Foundation. *Production Quality, Multilayer Open Virtual Switch* [online]. The Linux Foundation. [cit. 7. 4. 2020]. Dostupné z: <https://www.openvswitch.org/>.

[76] Tomáš Hlava. *Fáze a úrovně provádění testů* [online]. Portál zabývající se problematikou ověřování kvality software. [cit. 3. 4. 2020]. Dostupné z: <http://testovanisoftwaru.cz/category/druhy-typy-a-kategorie-testu>.

[77] V. Stroele, J. Oliveira, G. Zimbrao, and J.M. Souza. *Mining and analyzing mul- tirelational social networks.* Computational Science and Engineering, International Conference on, volume 4, 2009.

[78] VASSEUR, J. – ROUX, J. L. Path Computation Element (PCE) Communication Protocol (PCEP). RFC 5440 (Proposed Standard), March 2009. Dostupné z: <http://www.ietf.org/rfc/rfc5440.txt>.

[79] Vidya B. Harkal and Aaradhana A. Deshmukh. *Software Defined Networking with Floodlight Controller* [online]. IoT 2016 - Computer Science. [cit. 11. 4. 2020]. Dostupné z: <https://api.semanticscholar.org/CorpusID:17563926>.

[80] Yi Tseng. *Awesome SDN* [online]. Taiwan Software Defined Network Developer Society. [cit. 26. 1. 2020]. Dostupné z: <https://github.com/sdnds-tw/awesome-sdn>.

Příloha A

Seznam použitých zkratek

ACL Access Control List

AHP Analytic Hierarchy Process

API Application Programming Interface

BGP Border Gateway Protocol

BGP-LS Border Gateway Protocol Link-State

CLI Command Line Interface

CSS Cascading Style Sheets

DDoS Distributed Denial of Service

DNAC Digital Network Architecture Center

FAT Factory Acceptance Tests

FoRCES Forwarding and Control Element Separation

GUI Graphical User Interface

HA High Availability

HDCP Dynamic Host Configuration Protocol

HTML Hypertext Markup Language

HTTP/S HyperText Transfer Protocol/Secure

HW Hardware

I/O Input/Output

I2RS Interface to Routing System

IaaS Infrastructure as a Service

IBN Intent-Based Networking

InPs Infrastructure Providers

IP Internet Protocol

IPS Intrusion Prevention Systems

IS-IS Intermediate System to Intermediate System

ISE Identity Services Engine

ISO/OSI International Organization for Standardization/Open Systems Interconnection model

ISP Internet Service Provider

IT Information Technology

IVS Indigo Virtual Switch

JSON JavaScript Object Notation

JVM Java Virtual Machine

LAN Local Area Network

LINC Link Is Not Closed

LSPs Labeled Switched Paths

MAC Media Access Control

MCDM Multi-Criteria Decision Making

MIB Management Information Base

MPLS Multiprotocol Label Switching

MVC Model, View, Controller

NAT Network Address Translation

NE Network Element

NEC Nippon Electric Corporation

NETCONF Network Configuration Protocol

NFV Network Functions Virtualization

NGFWs Next-generation firewall

NMS Network Management Station

NTT Nippon Telegraph and Telephone Public Corp.

NVE Network Virtualization Environment

OF-Config OpenFlow-Config

OMG Object Management Group

ONF Open Networking Foundation

OSGi Open Services Gateway initiative

OSGi Open Services Gateway initiative

OSPF Open Shortest Path First

OSS Operational Support System

OVSDB Open vSwitch Database Management Protocol

PaaS Platform as a Service

PCE Path Computation Element

PCEP Path Computation Element Communication Protocol

PHP Hypertext Preprocessor

REST Representational State Transfer

REST Representional State Transfer

RPC Remote Procedure Call

RPC Remote Procedure Calling

SaaS Software as a Service

SD-Access Software-Defined Access

SD-Branch Software-Defined Branch

SD-WAN Software-Defined Wide Area Network

SDN Software Defined Networking

SLA Service Level Agreement

SNMP Simple Network Management Protocol

SPOF Single point of failure

SPs Service Providers

SSH Secure Shell

SUT System Under Test

SVF Super Virtual Fabric

SW Software

TCP Transmission Control Protocol

TED Traffic Engineering Database

TLS Transport Layer Security

UC Use Case

UML Unified Modeling Language

URL Uniform Resource Locator

VLAN Virtual Local Area Network

VPN Virtual Private Network

VPP Vector Packet Processing

VXLAN Virtual Extensible LAN

WAN Wide Area Network

XML Extensible Markup Language

XMPP Extensible Messaging and Presence Protocol

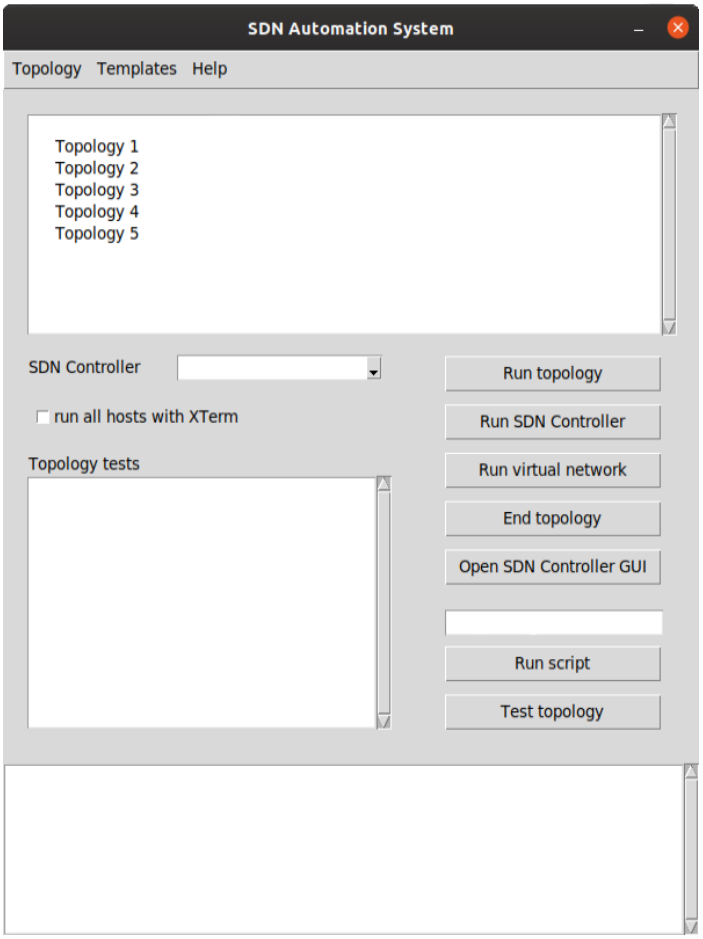
Příloha B

Obsah přiloženého archivu

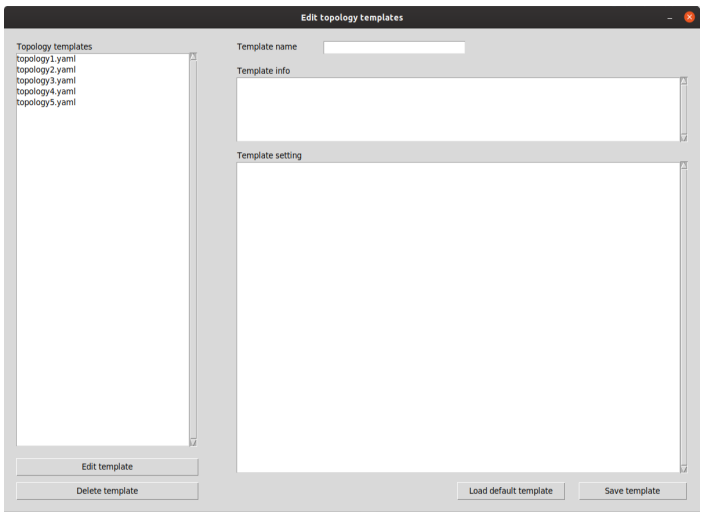
- /
  - `Text` ..... text diplomové práce
    - `DP_Flachs_2020.pdf` ..... diplomová práce ve formátu PDF
    - `DP_Flachs_2020.tex` ..... diplomová práce ve formátu TEX
    - `DP_Flachs_2020_exercises.pdf` ..... praktická cvičení ve formátu PDF
    - `DP_Flachs_2020_exercises.tex` ..... praktická cvičení ve formátu TEX
  - `SDN Automation System` ..... systém pro automatizaci výuky
    - `Zdrojové kódy` ..... zdrojové kódy
      - `Systém` ..... zdrojové kódy systému
      - `Praktická cvičení` ..... zdrojové kódy praktických cvičení
    - `Dokumentace` ..... dokumentace systému

Příloha C

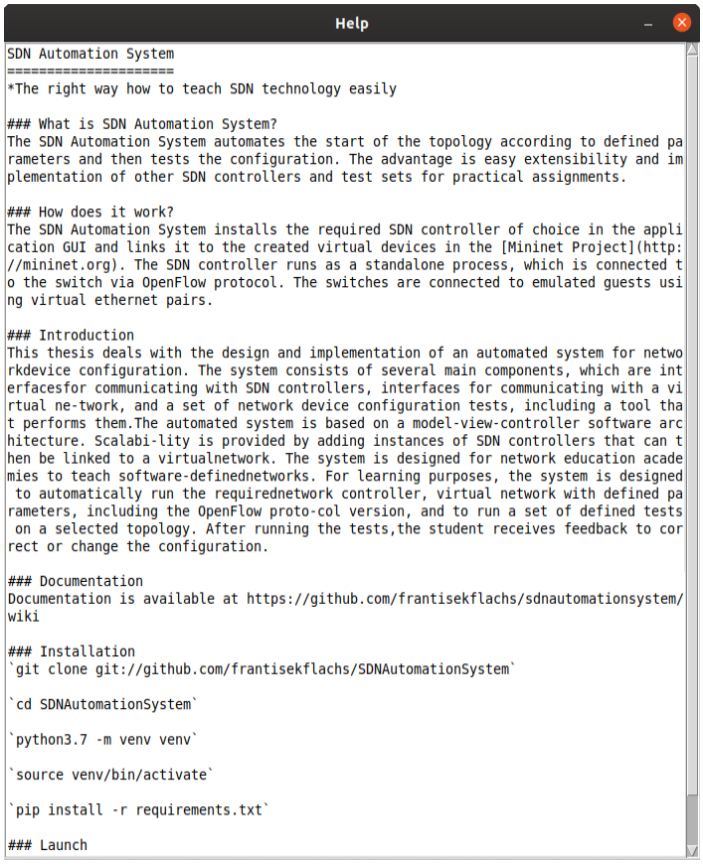
Grafické rozhraní systému



Obrázek C.1: GUI - Okno pro správu topologie



Obrázek C.2: GUI - Okno pro správu šablon topologie



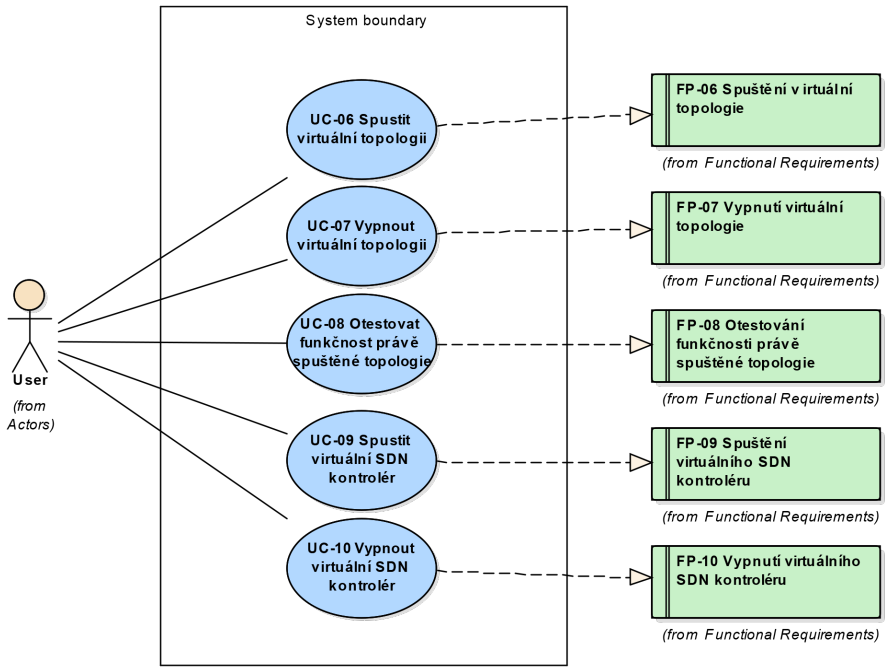
Obrázek C.3: GUI - Okno pro zobrazení nápovědy

Příloha D

Modely a diagramy systému

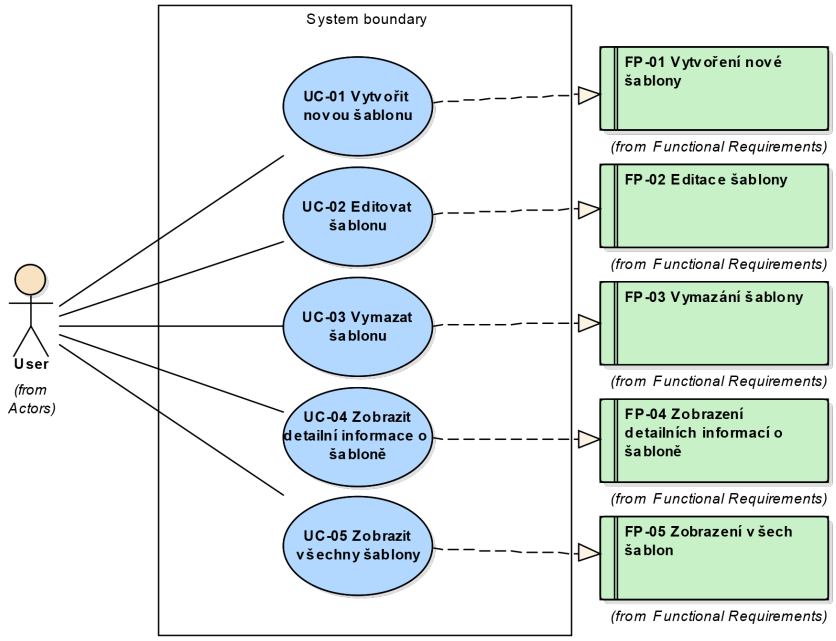
D.1 Modely případů užití

D.1.1 Topologie sítě



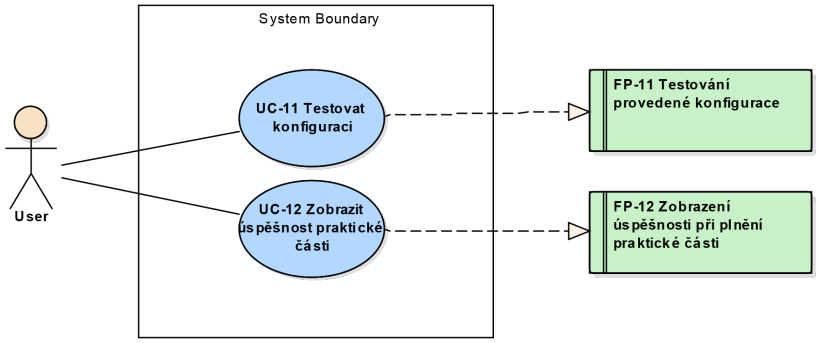
Obrázek D.1: Model případu užití topologie sítě

D.1.2 Šablona topologie



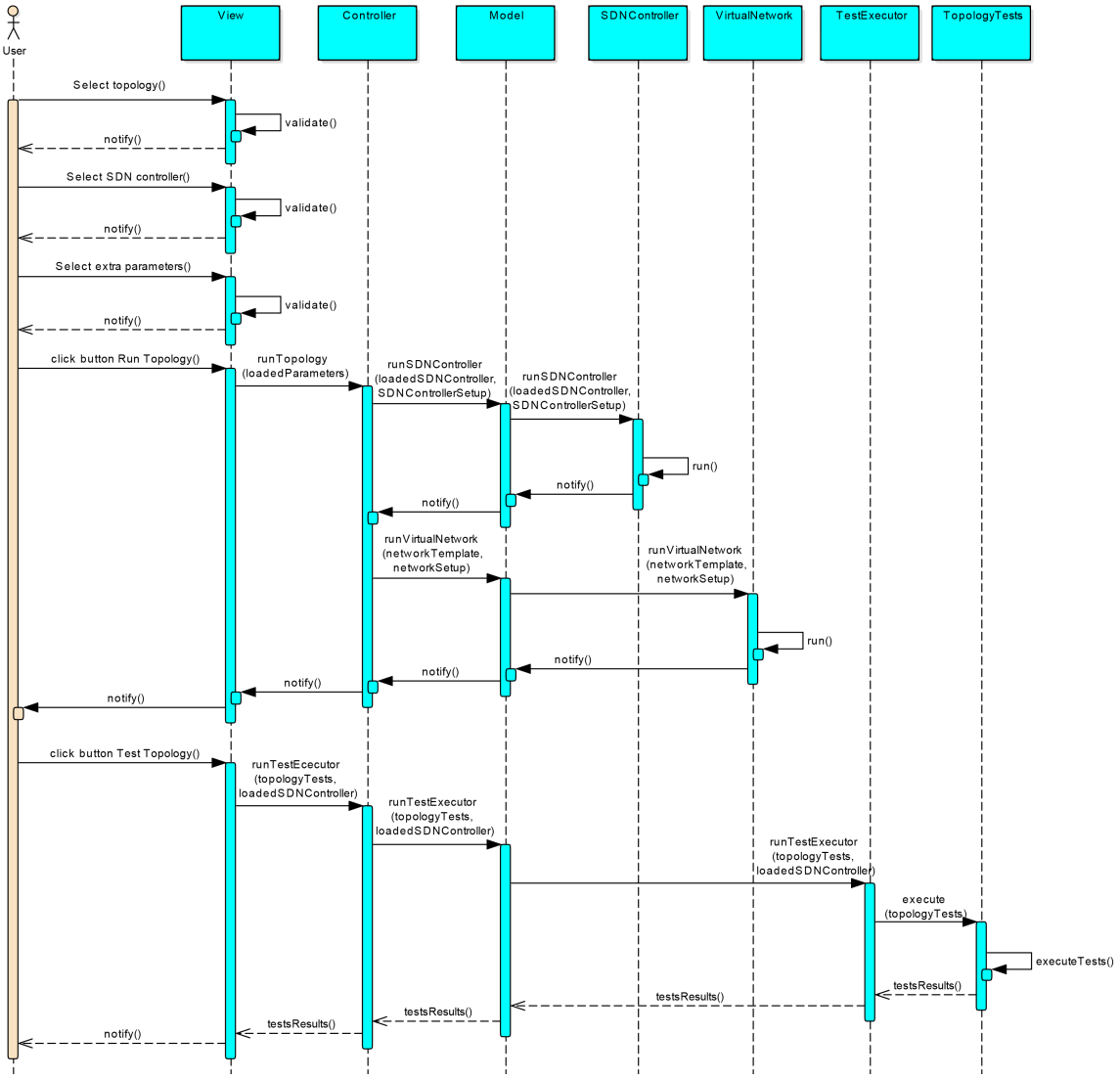
Obrázek D.2: Model případu užití pro práci s šablonami

D.1.3 Praktická zadání



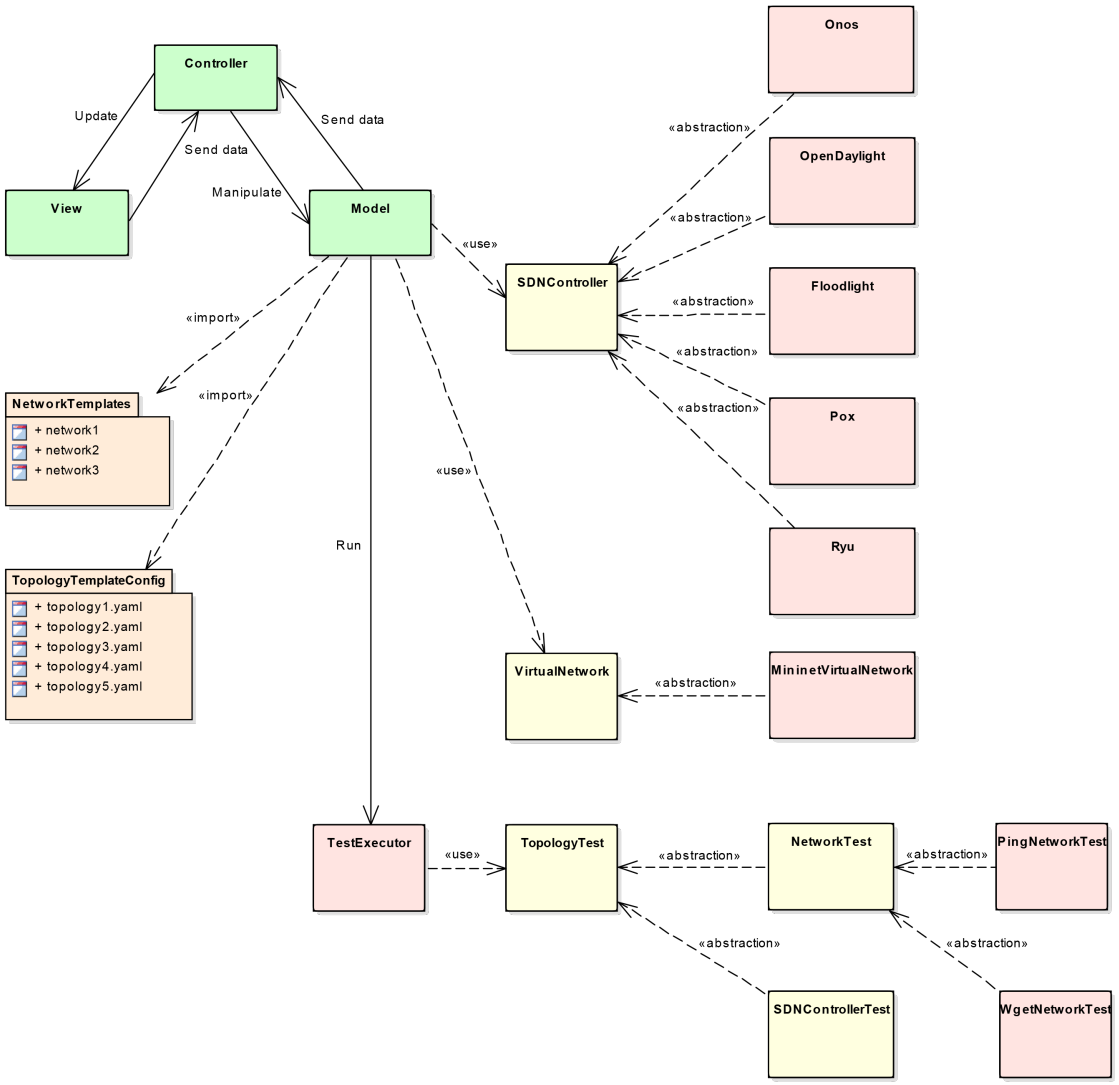
Obrázek D.3: Model případu užití pro praktická zadání

D.2 Sekvenční diagram spuštění a testování topologie



Obrázek D.4: Sekvenční diagram spuštění a testování topologie

D.3 Diagram tříd systému



Obrázek D.5: Diagram tříd systému

Příloha E

Rozšíření systému a vlastní implementace

Systém je možné snadno rozšířit o nová zadání zahrnující šablonu topologie, virtuální síť a sadu testů. Další rozšiřitelnost je z pohledu implementace nového SDN kontroléru.

E.1 Nové zadání

Vytvořit lze libovolné zadání, které bude simulovat problematiku související jak s klasickou sítí, kde přepínače fungují s řídicí a datovou vrstvou, tak s přepínači komunikujícími s SDN kontrolérem.

E.1.1 Šablona topologie

V ukázce E.1 je výchozí šablona topologie. Tato šablona je načtena v okně pro správu šablon topologie po stisknutí tlačítka *Load default template* (viz kapitola 3.14.2). Současně je možné tuto šablonu použít pro tvorbu nové topologie vložením kódu do nově vytvořeného souboru s příponou *yaml*. Pokud je soubor obsahující šablonu topologie umístěn v adresáři */SDNAutomationSystem/topology_templates_config/*, bude šablona automaticky načtena při dalším spuštění systému.

```
### TOPOLOGY DESCRIPTION ###
topologyDescription:
topologyAuthor:

topologyVersion:
topolofyOFVersion:

### TOPOLOGY SETUP ###
topologyTests:

networkTemplate:

networkSetup:
```

```
14
15 ### SDN CONTROLLERS SETUP  ###
16 Floodlight:
17
18 Onos:
19
20 Opedaylight:
21
22 Pox:
23
24 Ryu:
25
26 ### SDN CONTROLLERS POST CONFIGURATION ###
27 sdnControllersPostConfig:
```

Seznam zdrojových kódů E.1: Šablona topologie

E.1.2 Virtuální síť v Mininetu pomocí API

V ukázce E.2 je znázorněna šablona pro tvorbu nové virtuální sítě. Současně je možné tuto šablonu použít pro tvorbu virtuální sítě vložením zdrojového kódu do nově vytvořeného souboru s příponou `.py`. Pokud je soubor obsahující šablonu virtuální sítě umístěn v adresáři */SDNAutomationSystem/network_templates/*, je možné na něj odkazovat ze šablony topologie.

```
1  from mininet.topo import Topo
2
3  class MyNetwork(Topo):
4
5      def build(self):
6
7          # here define hosts, switches, links and controllers
8
9          # example:
10
11         # host1 = self.addHost('h1')
12         # switch1 = self.addSwitch('s1')
13         # self.addLink(host1, switch1)
14
15 topos = {'mynetwork': (lambda: MyNetwork())}
```

Seznam zdrojových kódů E.2: Šablona virtuální sítě

E.1.3 Testy topologie

V ukázkách E.3 a E.4 jsou definovány šablony pro tvorbu testů pro praktická zadání. Testy virtuální sítě jsou založeny na testování primárně síťových funkcionalit, které jsou povoleny/zakázány v důsledku konfigurace SDN kontroléru případně síťových zařízení. Testy SDN kontroléru jsou založeny na kontrole konfigurací funkcionalit samotného SDN kontroléru.

Pro tvorbu nových testů je nutné použít tyto šablony, kde každá z nich implementuje abstraktní třídu definovanou modelem systému. Pro testy virtuální sítě je to abstraktní třída

NetworkTest a pro testy SDN kontroléru to je třída *SDNControllerTest*. Pro používání testu je nutné, aby byl umístěn do správného adresáře.
virtuální síť: */SDNAutomationSystem/topology_tests/network_tests/*
SDN kontrolér: */SDNAutomationSystem/topology_tests/controller_tests/*

```
from topology_tests.network_tests.network_test import NetworkTest

class MyTest1(NetworkTest):

    def execute(self, params):
        try:
            # test code here
            return True
        except:
            return False
```

Seznam zdrojových kódů E.3: Šablona testu virtuální sítě

```
from topology_tests.controller_tests.sdn_controller_test import
    SDNControllerTest

class MyTest2(SDNControllerTest):

    def execute(self, params, SDNController):
        try:
            # test code here
            return True
        except:
            return False
```

Seznam zdrojových kódů E.4: Šablona testu SDN kontroléru

Posledním nezbytně nutným krokem je přidání reference na nově implementovaný test, pod kterým bude test odkazován ze šablony do seznamu testů ve třídě *TestExecutor* v souboru *test_executor.py* (viz zdrojový kód E.5). Jedná se o mapování názvu testu na třídu, ve které je test definován.

```
from topology_tests.network_tests.my_test import MyTest1
from topology_tests.controller_tests.sdn_controller_test import MyTest2

class TestExecutor:

    def __init__(self, SDNController):
        ...
        self.implementedTests = {
            ...
            'my_test1': MyTest1(),
            'my_test2': MyTest2()
        }
        ...
```

Seznam zdrojových kódů E.5: Přidání testu do třídy TestExecutor

E.1.4 Post konfigurace

V šabloně topologie je možné definovat odkaz na konfigurační skript, který proběhne po spuštění a zprovoznění topologie. Post konfigurační skript je možné definovat následujícím způsobem `sdnControllersPostConfig: "topologyXPostConfig"`. Skripty jsou umístěné v adresáři `post_configurations`.

Po výběru SDN kontroléru v GUI a stisknutím tlačítka pro spuštění topologie nebo spuštění kontroléru se zapne vybraný SDN kontrolér. Následně se spustí post konfigurační skript. Skript se pokusí připojit ke kontroléru pomocí REST API a zkontrolovat jeho funkčnost. Pokud je SDN kontrolér funkční a připraven ke komunikace pomocí REST API, jsou provedeny následně definované kroky, které jsou navázány na dané praktické zadání. Příkladem pro post konfigurační skript může být zapnutí některé z funkcionalit kontroléru, případně vložení OpenFlow toků pro konkrétní síťové zařízení.

```
class PostConfigTemplate:
    """Topology post configuration template"""

    def execute(self, sdnc):
        """Execute post configuration"""

        try:
            sleep(5)
            if not sdnc.isRunning():
                print('SDN Controller is not running.')
                return False
            else:
                print('Configuring Post Config for ' + str(sdnc))

            # loaded controller is Floodlight
            if isinstance(sdnc, Floodlight):
                ret = self.floodlightConfig(sdnc)

            # loaded controller is Onos
            elif isinstance(sdnc, Onos):
                ret = self.onosConfig(sdnc)

            # loaded controller is Opendaylight
            elif isinstance(sdnc, Opendaylight):
                ret = self.opendaylightConfig(sdnc)

            # loaded controller is Pox
            elif isinstance(sdnc, Pox):
                ret = self.poxConfig(sdnc)

            # loaded controller is Ryu
            elif isinstance(sdnc, Ryu):
                ret = self.ryuConfig(sdnc)

            return ret

        except Exception as e:
            print("Something went wrong " + str(e))
            return False
```

```
40
41      def floodlightConfig(self, sdnc):
42          """Configuring Floodlight controller
43          after started for this example"""
44
45          try:
46              # do SOME POST CONFIG
47              return True
48          except Exception as e:
49              print("Something went wrong " + str(e))
50              return False
51
52      ...
```

Seznam zdrojových kódů E.6: Šablona post konfiguračního skriptu

E.2 Přidání SDN kontroléru

V ukázce E.7 je znázorněno, jakým způsobem je nutné implementovat rozhraní pro nově implementovaný SDN kontrolér. Je nutné použít abstraktní třídu *SDNController*. Rozhraní pro SDN kontroléry jsou umístěna v adresáři */SDNAutomationSystem/sdn_ controllers/*.

```
1  from sdn_controllers.sdnController import SDNController
2
3
4  class MySDNController(SDNController):
5
6      def run(self, SDNControllerSetup):
7
8      def showSDNControllerGui(self):
9
10     def addFlow(self, data):
11
12     def deleteFlow(self, data):
13
14     def listFlowTable(self, device):
15
16     def clearFlowTable(self, device):
17
18     ...
```

Seznam zdrojových kódů E.7: Přidání rozhraní pro nový SDN kontrolér

Posledním nutným krokem je přidání nového kontroléru do seznamu implementovaných SDN kontrolérů v souboru *config.py*, což je znázorněno v ukázce E.8.

```
1  from sdn_controllers.my_sdn_controller import MySDNController
2
3  ...
4
5  implementedSDNControllers = {
6      ...
```

```
7       'MySDNController': MySDNController
8   }
9       ...
```

Seznam zdrojových kódů E.8: Přidání nového SDN kontroléru do seznamu implementovaných kontrolérů

Příloha F

Poznámky z testování a výběru vhodných komponent systému

F.1 Automatizovaný systém

- Jelikož automatizovaný systém používá většinu nástrojů, které ovlivňují konfiguraci podkladového systému Linux, je nutné ho spouštět s příslušným oprávněním. Pro vývoj SW byl vybrán nástroj PyCharm. Aby bylo možné spouštět a ladit vyvíjený systém z tohoto nástroje, musel být PyCharm spouštěn také s příslušným oprávněním. Tento způsob spouštění často přinesl mnohdy komplikace pro uživatele root.
- Při vývoji systému jsem narážel na problémy s programem PyCharm, který kolikrát doslova zamrzl, pokud byl spuštěn z CLI.
- Pro spuštění CLI okna pro každý nástroj, který je definován v topologii, je nutné spustit samostatný proces *gnome-terminal*. Ukázalo se, že předávání PID procesu není triviální záležitostí. Řešení je v implementaci třídy konkrétního kontroléru.
- Při používání automatizovaného systému pro různá zadání se opakovaně spouští různé virtuální instance, díky čemuž se pravidelně mění veřejné klíče, které jsou uloženy v souboru `~/.ssh/known_hosts`. Aby bylo možné se k hostům pomocí SSH připojit bez nutnosti stálého obnovování klíčů, byla tato bezpečnostní funkcionalita vypnuta. Jelikož není nutné komunikovat z VM do internetu, nehrozí tak útok *man-in-the-middle*. Případnou aktualizaci veřejného klíče hosta je možné provést zadáním příkazu `ssh-keygen -f "home/user/.ssh/known_hosts" -R "{host_ip_address}"`, kde proměnnou `{host_ip_address}` nahradíme IP adresou hosta.

F.2 SDN kontroléry

- Nemá smysl vytvářet „prodlouženou ruku“ pro SDN kontrolér ve smyslu duplikace grafického rozhraní. Pokud SDN kontrolér nemá přímo vestavěné grafické rozhraní, je vhodné najít jiný projekt, který by kontrolér doplňoval, případně použít pro práci s kontrolérem jedno z dostupných API.

- DNA Center nepodporuje funkcionalitu zálohování a obnovení konfigurace pomocí REST API. Zálohování je možné pouze pomocí GUI. Výhledy rozvoje REST API a jeho dokumentaci zatím nejsou k dispozici.
- Dokumentace u SDN kontrolérů není dostatečně podrobná. Některé verze, které jsou uvedeny jako stabilní, není možné spustit bez nainstalování balíčků a nástrojů, které nejsou řádně popsané v dokumentaci kontrolérů. Nutné balíčky na instalaci jsou uvedeny v kapitole 3.12.2.
- pro sestavení programu SDN kontroléru, který je napsán v jazyce Java, je nutné nainstalovat správnou verzi Javy, kterou daný SDN kontrolér vyžaduje. Verze Javy je různá napříč kontroléry napsanými v jazyce Java.
- Kontrolér Ryu se mi nepovedlo nainstalovat pomocí PIP3 (Python 3.7, Ubuntu 18.04) z důvodu neznámé chyby ve zdrojovém kódu. Bylo tedy nutné stáhnout data z repozitáře na GitHubu a projekt sestavit ručně. Po sestavení se mi následně nepovedlo Ryu spustit z vytvořeného adresáře v definované cestě pro build. Bylo tedy nutné spustit příkaz *python ./setup.py install*. Následně se vytvořily adresáře */bin/ryu-manager a /bin/ryu*, ze kterých je možné Ryu kontrolér spustit a pracovat s ním.
- Pro kontrolér Ryu verze 4.34 není funkční balíček *gui_topology.py* spolu s balíčkem *simple_switch_15.py*. Což znamená, že GUI kontroléru vykazuje chyby pro OpenFlow verzi 1.5.
- Kontrolér Floodlight ve verzi 1.2 má chybu v dokumentaci pro nástroj *staticentrypusher*, který se přejmenoval z *staticflowpusher*. Jelikož se jméno nástroje objevuje v URL volání REST API, je nutné správně uvést cestu.
- Kontrolér Floodlight ve verzi 1.2 správně nezobrazuje informace ve svém GUI při použití protokolu OpenFlow verze 1.5.
- Kontrolér Floodlight sice podporuje velké množství verzí protokolu OpenFlow, avšak má problém porozumět zprávě *hello*, kterou si vyměňuje kontrolér s přepínačem. Neporozumění této zprávě způsobí chybu, což následně vede k neustálému připojování a odpojování síťového zařízení. Toto chování lze upravit specifikací konkrétní verze protokolu na síťovém zařízení, kterou chceme použít pro navázání komunikace.
- Kontrolér POX nepracuje správně s přepínačem `--mac` pro nástroj Mininet, způsobuje konflikt MAC adres.
- Kontrolér POX pro spuštění vyžaduje pouze Python verze 2.7. V ostatní verzích je nestabilní nebo se nespustí vůbec.
- Pro kontrolér POX není k dispozici GUI. Avšak existuje knihovna, pomocí které lze propojit kontrolér POX s open-source nástrojem Gephi [26]. Tento nástroj je použit pro zobrazení dat z kontroléru. Pro zprovoznění tohoto nástroje je nutné manuálně definovat cestu k `\$JAVA_HOME`, protože není zajištěna kompatibilita s novější verzí Javy, než verzí 1.8.
- Kontrolér ONOS správně funguje pouze pokud je umístěn v cestě */opt/onos/*.

- Konzole kontroléru ONOS a OpenDaylight nepracuje správně s xterm-256color v proměnné TERM. Pro správnou funkčnost interaktivní konzole je nutné změnit hodnotu této proměnné například: `export TERM=xterm-color`.

- Pro kontrolér OpenDaylight není zveřejněná dokumentace pro REST API, které je dostupné pomocí RESTCONF. Přehled všech metod volání je však možné zobrazit pomocí URL adresy uvedené v tabulce G.5. Ne všechny uvedené metody jsou však implementované v RPC (Remote Procedure Call) pro danou verzi kontroléru. Pro zjištění, které metody je potřeba využít, bylo nutné použít debugovací nástroj v prohlížeči Mozilla Firefox, pomocí které je možné zobrazit aktuální komunikaci webové aplikace. K debugování REST volání byl použit nástroj OpenDaylight-Openflow-App. Jeho instalace je popsaná v 3.4

F.3 Open-source projekt Mininet

- Zásadní problém, který jsem musel řešit, je nekompatibilita zdrojového kódu Mininet, který je napsán v Python 2, s SDN Automation System, který jsem se rozhodl psát v Python 3. I když je v dokumentaci projektu Mininet uvedena kompatibilita s verzí Python 3, ne všechny třídy a metody jsou upraveny. Bylo tedy nutné zasáhnout do tohoto zdrojového kódu a ty třídy, které bylo potřeba využít, upravit pro kompatibilu s Python 3.
  Například úpravy ve třídě Topo:

```
1 class Topo( object ):
2
3 Python 2: def __init__(self, args: object, params: object) -> object:
4 Python 3: def __init__(self, *args, **params):
5
6 Python 2: def addSwitch(self, name: object, opts: object) -> object:
7 Python 3: def addSwitch(self, name, **opts):
```

Seznam zdrojových kódů F.1: Úprava kódu z Python 2 do Python 3 v projektu Mininet

- Pokud chceme spustit příkaz v CLI daného virtuálního hosta spouštěného pomocí projektu Mininet, je nutné použít API nejvyšší úrovně popsané v dokumentaci. Mininet je možné spustit z CLI pomocí příkazu *sudo mn*. Tento přístup bohužel nepodporuje automatizovaně spustit příkaz na klientech následně po jejich spuštění.

- Pro virtualizaci přepínače jsem použil nástroj Open vSwitch. Mininet přikládá možnost nainstalovat tento nástroj spolu s projektem Mininet. Avšak v dokumentaci již není uvedeno, o jakou verzi se jedná. Jelikož OpenFlow od verze 1.31 je podporován v OVS až od verze 2.2, bylo nutné stáhnout a sestavit tento projekt samostatně.

- Před spuštěním nového procesu Mininet spolu s virtuálními zařízeními je vhodné spustit nejprve *mn –clean*. Pokud je předchozí proces ukončen nesprávným způsobem, případně je použitý příkaz *sudo kill PID*, pak není odstraněna všechna nastavená konfigurace a při dalším pokusu o spuštění systému dojde k chybě.

Příloha G

Definice základních REST volání pro SDN kontroléry

SDNController	**SDNController::Floodlight**
run(self, SDNControllerSetup):	java -jar floodlight.jar
showSDNControllerGui(self):	http://localhost:8080/ui/index.html
addFlow(self, data):	restCall('/wm/staticentrypusher/json', data, 'POST')
deleteFlow(self, data):	restCall('/wm/staticentrypusher/json', data, 'DELETE')
listFlowTable(self, device):	restCall('/wm/staticentrypusher/list/ {device}/json', {}, 'GET')
clearFlowTable(self, device):	restCall('/wm/staticentrypusher/clear/ {device}/json', {}, 'GET')
Dokumentace	<`https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343539/Floodlight+REST+API`>

Tabulka G.1: Floodlight REST volání

SDNController	**SDNController::POX**
run(self, SDNControllerSetup):	pox.py [params]
showSDNControllerGui(self):	http://localhost:8000/
addFlow(self, data):	restCall('/OF/', {"method": "set_table", "params":{}, "id": 0}, 'POST')
deleteFlow(self, data):	-
listFlowTable(self, device):	restCall('/OF/pretty/', {"method": "get _flow_stats", "params": {"dpid": device}, "id": 0}, 'POST')
clearFlowTable(self, device):	-
Dokumentace	<`https://noxrepo.github.io/pox-doc/html/`>

Tabulka G.2: POX REST volání

SDNController	**SDNController::Ryu**
run(self, SDNControllerSetup):	/bin/ryu-manager [params]
showSDNControllerGui(self):	http://localhost:8080/
addFlow(self, data):	restCall('/stats/flowentry/add', data, 'POST')
deleteFlow(self, data):	restCall('/stats/flowentry/delete', data, 'POST')
listFlowTable(self, device):	restCall('/stats/flow/{device}', {}, 'GET')
clearFlowTable(self, device):	restCall('/stats/flowentry/clear/{device}', {}, 'DELETE')
Dokumentace	<`https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html`>

Tabulka G.3: Ryu REST volání

SDNController	**SDNController::ONOS**
run(self, SDNControllerSetup):	onos-service start
showSDNControllerGui(self):	http://localhost:8181/onos/ui/
addFlow(self, data):	restCall('/onos/v1/flows?appId=666', data, 'POST')
deleteFlow(self, data):	restCall('/onos/v1/flows/{device}/{flow}', 'DELETE')
listFlowTable(self, device):	restCall('/onos/v1/flows/{device}', {}, 'GET')
clearFlowTable(self, device):	-
Dokumentace	<`https://wiki.onosproject.org/display/ONOS/Appendix+B%3A+REST+API`>

Tabulka G.4: ONOS REST volání

SDNController	**SDNController::OpenDaylight**
run(self, SDNControllerSetup):	./bin/karaf
showSDNControllerGui(self):	http://localhost:8181/index.html, http://localhost:9000/
addFlow(self, data):	restCall('/restconf/config/opendaylight -inventory:nodes/node/{device}/table/ {tableId}/flow/{flowId}', data, 'PUT')
deleteFlow(self, data):	restCall('/restconf/config/opendaylight -inventory:nodes/node/{device}/table/ {tableId}/flow/{flowId}', data, 'DELETE')
listFlowTable(self, device):	restCall('/restconf/operational/opendaylight-inventory:nodes/node/{device}', {}, 'GET')
clearFlowTable(self, device):	-
Dokumentace	<`http://localhost:8181/index.html#/yangui/index`>

Tabulka G.5: OpenDaylight REST volání