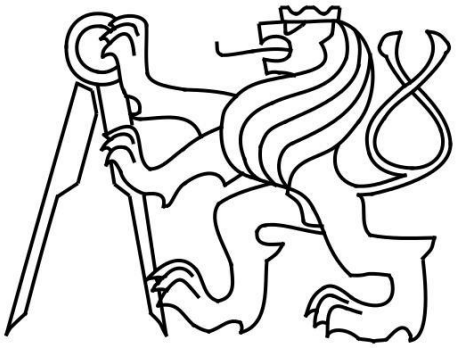
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická
Katedra telekomunikační techniky



Diplomová práce

Laboratorní platforma softwarově definovaných datových sítí

Laboratory platform for software defined networks

Duben 2019

Diplomant: Bc. Matěj Jehlička

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

Čestné prohlášení

Prohlašuji, že jsem zadanou diplomovou práci vypracoval samostatně a že jsem v seznamu použité literatury uvedl všechny prameny, z kterých jsem vycházel. Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé diplomové práce nebo její části se souhlasem katedry.

V Praze dne …….. ……..…….

Podpis diplomanta

Poděkování

Rád bych poděkoval vedoucímu práce doc. Ing. Leoši Boháčovi, Ph.D., za odborné vedení, rady a připomínky k práci. Dále bych chtěl poděkovat mé rodině a přátelům za vytrvalou podporu při řešení této práce.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Jehlička** Jméno: **Matěj** Osobní číslo: **475403**

Fakulta/ústav: **Fakulta elektrotechnická**

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

Studijní program: **Elektronika a komunikace**

Studijní obor: **Komunikační systémy a sítě**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Laboratorní platforma softwarově definovaných datových sítí

Název diplomové práce anglicky:

Laboratory Platform for Software Defined Networks

Pokyny pro vypracování:

Navrhněte laboratorní platformu pro testování a výuku konceptu softwarově definovaných sítí (SDN) s přesahem do konceptu virtualizace síťových funkcí (NFV).
Cílem práce bude:
1) zhodnocení současného stavu rozvoje konceptů SDN a NFV
2) zhodnocení současných technických řešení umožňujících implementaci SDN a NFV
3) návrh laboratorní platformy konceptu SDN a NFV určené pro výuku a testování nových síťových aplikací
4) implementace navržené platformy a vytvoření základních scénářů její funkce
5) návrh několika vzorových síťových aplikací určených pro experimenty a výuku v oblasti SDN a NFV
6) zhodnocení významu SDN pro budoucí praxi

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 978-1491931257.
[2] NADEAU, Thomas D. a Kenneth GRAY. SDN: software defined networks. Beijing: O'Reilly, [2013]. ISBN 978-1449342302.
[3] CHAYAPATHI, Rajendra. Network function virtualization (nfv) with a touch of sdn. Indianapolis, IN: Addison-Wesley Professional, 2016. ISBN 978-0134463056.

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: **25.01.2019** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **20.09.2020**

doc. Ing. Leoš Boháč, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. 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

Anotace

Práce se zabývá současným stavem softwarově definovaných sítí. Analyzovali jsme nejčastější problémy v sítích a jejich možná řešení pomocí SDN. Uvádíme shrnutí dostupných hardwarových a softwarových platforem a příklady jejich vhodných použití v souladu s výhodami SDN. Práce se dále soustředí na návrh a realizaci dostupného testovacího pracoviště, které nalézá svoje primární uplatnění v prvním seznámení a výuce v oblasti SDN a NFV. Následují kapitola nabízí vypracované vzorové laboratorní úlohy pro výuku SDN a NFV.

Klíčová slova

SDN, laboratorní platforma, *RaspberryPi*, *Open vSwitch*

Summary

The diploma thesis evaluates current state of software defined networking. We have analyzed common challenges in networking and possible solutions provided by SDN. It creates a summary of available hardware and software platforms and examples of use according to the benefits of SDN. The thesis also focuses on the design and implementation of an affordable test bed, which finds it's main use as a teaching tool for the first steps in learning SDN and NFV. Next chapter includes laboratory assignments for learning SDN and NFV.

Keywords

SDN, test bed, *RaspberryPi*, *Open vSwitch*

Obsah

Úvod

Práce se zabývá současným stavem a možnostmi nasazení SDN (softwarově definovaných sítí). Tento inovativní přístup k architektuře sítí vychází z přirozeného vývoje informačních technologií a umožňuje další vývoj komponent potřebných pro skupiny služeb označovaných „X jako služba“. Mezi ty patří například virtualizace a řetězení síťových funkcí, „*multi-tenant cloud*“, potlačování útoků DDoS (*distributed denial of service*), virtualizace jádra mobilní sítě, „*mobile edge computing*“ aj. SDN je tak důležitým stavebním kamenem pro efektivnější řízení sítí nejen v datových centrech, ale i v optických WAN (*wide area network*) sítích, či sítích mobilních operátorů dle nového standardu 5G.

Práce udává ucelený přehled o stavu problematiky softwarově definovaných sítí. Popisuje relevantní a nejčastěji řešené výzvy v datových sítích a návrhy jejich současných řešení pomocí SDN. Dále předkládá využívané softwarové i hardwarové prostředky pro jejich realizaci. Mezi nimi uvádíme dostupné přepínače, kontroléry, programovací jazyky, či síťové operační systémy.

Vzhledem k nedostatku snadno dostupných testovacích platforem umožňujících si v rámci SDN otestovat napsaný program, či proprietární protokol, práce popisuje vlastní návrh a implementaci levné a otevřené platformy. Ta má za úkol přinést názornou a volně přístupnou didaktickou pomůcku pro rozšíření práce a experimentů s SDN v rámci laboratorních cvičení věnovaným výuce SDN. Díky využití masově rozšířeného hardwaru a nízké pořizovací ceně, je takto navržená platforma vhodná i pro domácí přípravu na výuku, či pro experimenty a samostatnou práci studentů mimo školu.

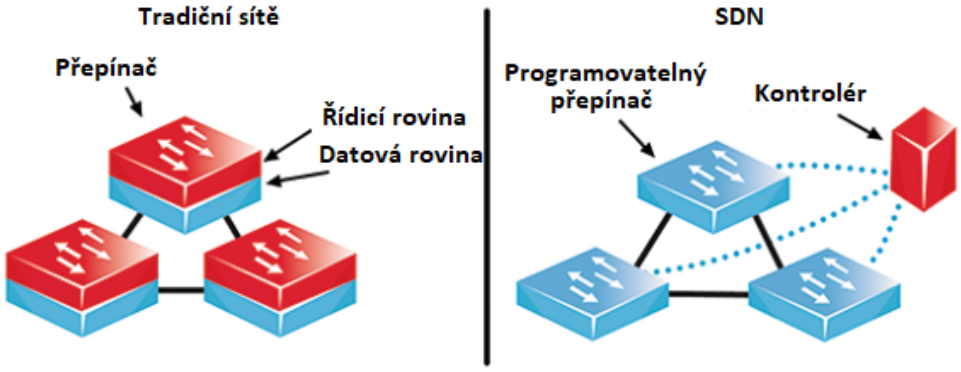
V neposlední řadě přinášíme podpůrné materiály ve formě vzorových úloh, které mají sloužit jako podklad pro práci studentů na laboratorních cvičeních. Těmi se snažíme podpořit praktickou výuku SDN v rámci školních laboratoří. Mimo teoretického úvodu a podrobného postupu obsahují také verzi pro vyučujícího. Ta obsahuje definované cíle úloh, potřebnou přípravu, časovou náročnost nebo také nejčastější problémy, které mohou během vypracování úloh nastat.

1 Softwarově definované sítě

1.1 Základní pojmy

SDN (*Software defined network*) je nový přístup k architektuře, který slibuje větší dynamiku, flexibilitu a schopnost adaptovat se na současný síťový provoz. [1]

Klasický síťový prvek se obecně skládá ze 2 rovin, z řídicí a datové. Řídicí rovina se stará o sledování a řízení toků paketů, které skutečně zpracovává nižší, datová rovina. V SDN dochází k oddělení datové roviny síťového prvku od řídicí roviny, která tak může být přístupná k úpravám a přímo ovlivnitelná přes tzv. API (*Application Programming Interface*). Nabízí tak možnost síťovému administrátorovi zasahovat do řízení toků paketů. Díky centralizaci řídicí roviny a globálnímu přehledu o stavu sítě lze efektivněji řídit datový provoz a přizpůsobovat síťové parametry na základě různorodých požadavků. Mezi ty nejčastější patří zpoždění, fázové chvění, chybovost, přenosová rychlost, úroveň zabezpečení, dostupnost aj. [2]



Obrázek 1.1 - Rozdíl mezi klasickou architekturou a SDN, upraveno z [3]

Často zaměňovaným pojmem s SDN je **NFV (*Network Functions Virtualization*)**, se kterým se ve skutečnosti SDN navzájem doplňují. Díky NFV je možné vzít síťové služby (jako jsou například DHCP či DNS) a spustit je za pomocí kontejnerů jako snadno přenositelné virtualizované funkce. NFV poskytuje síťové funkce a SDN je řídí a efektivně jim zajišťuje konektivitu. [4] NFV se tak ve spolupráci s SDN podílí na vzniku celých virtualizovaných sítí (***Network virtualization***).

Sítě na základě bílých skříněk (*White box networking*) je trend oddělování softwarové a hardwarové části zařízení. To vede k možnosti nasazování hardwaru vhodného pro obecné použití, někdy také označované jako COTS (*Commercial Of The Shelf*). Ten je s výhodou používán nejčastěji ve formě výkonných serverů, nahrazujících drahé proprietární zařízení.

V případě přepínačů, někdy také označovaných jako „*bare-metal*“ přepínače, se využívá univerzálně využitelných čipových sad (*merchant-silicon chipset*), které přinášejí otevřenost řídicí roviny a s tím například možnost volby operačního systému.

Sítě založené na *„cloudu“ (Cloud networking)* jsou podobně jako u výpočtů v „*cloudu*“ snaha o posunutí správy, zabezpečení, či samotné konektivity do „*cloudu*“ a jejich následné poskytování jako služby.

1.2 Historie a vývoj

1.2.1 Přechod od klasických sítí

Klasické sítě zažívají posledních několik let příval kritiky. Narůstající provoz, jeho rozmanitost i potřeba řádného QOS (*Quality of Service*) posouvá současná řešení v podobě původní architektury k velmi vysokým cenám, malé škálovatelnosti a celkové složitosti. [5]

Dochází k čím dál rychlejšímu rozvoji služeb, které jsou často zaměřovány na konkrétní skupiny zájmu. S příchodem IOT (*Internet of Things*) a snahou o pokrytí každodenních činností chytrou elektronikou se zvyšují požadavky uživatelů na konektivitu „odkudkoliv, kamkoliv a kdykoliv“.

Mezi další významné faktory patří rozšíření ITaaS (*IT as a Service*), LTE (*Long Term Evolution*) a tím i zvyšující se mobilita uživatelů připojených k internetu. Rozmach virtualizace a technologií jako jsou například *Linuxové* kontejnery, umožňuje čím dál častější nasazování služeb v podobě virtuálních strojů. Takto virtualizované služby lze snáze migrovat a efektivněji tak využívat hardware na základě současného požadavku na výpočetní výkon. [6]

Sociální sítě umožňují neustálý kontakt lidí a změnu typu komunikace od „jeden k jednomu“ k „jeden k mnoha“. Narůstající počet uživatelů, kteří v čase produkují více dat a využívají více služeb vede k zajímavým příležitostem jak tato data dále zpracovávat a využívat je k optimalizaci byznysu. Potřeba uživatelská data sbírat a analyzovat dala za vznik velkým datům (*big data*).

Rapidní růst zaznamenávají *„cloudové“* technologie. *„Cloud“* tak využívají všichni od malých mobilních aplikací po celá datová centra. Podle studie *International Data Corporation* [7] bude alespoň polovina výdajů za IT směrem ke *„cloudovým“* řešením, dosáhne tak 60 % všech výdajů za IT infrastrukturu a 60 % až 70 % veškerých nákladů na služby, software a technologie do roku 2020.

CDN (*Content Delivery Network*) jsou významným prvkem pro práci s narůstajícím provozem. Jedná se o rozlehlé distribuované systémy serverů, které uživatelům dodávají požadovaný obsah. Servery jsou strategicky rozmístěny tak, aby byly co nejblíže k lokálním uživatelům, čímž zvyšují dostupnost a snižují zpoždění. [8]

Díky výše zmíněným skutečnostem tak dochází k neustále narůstajícímu rozdílu mezi požadavky trhu a relativně stagnujícími možnostmi tradičního řešení sítí.

1.2.2 Změna přístupu k návrhu

Čím dál častěji pohlížíme na síť jako na určitou platformu pro poskytování služeb. Dochází tak k přechodu na trend „*service oriented architecture*“, kdy se nebuduje nejdříve samotná infrastruktura, ale nahlíží se na projekt jako komplexní řešení daného byznys problému. To se pak postupně skládá z menších, zaměnitelných komponent, které jsou často otevřeným zdrojovým kódem (*open-source*).

Vznik SDN tak vychází z přirozeného vývoje softwaru, hardwaru, a zvyšujících se požadavků na sítě a jejich provozovatele. Lze tak pozorovat postupný trend virtualizace prostupujícího do sítí. V současnosti se k poskytování konektivity jako služby využívá protokol MPLS. NFV nabízí přidanou hodnotu v podobě hotových síťových služeb a SDN pak navrch přidává dosud chybějící flexibilitu. Pokud použijeme analogii silničního provozu, lze říci, že MPLS nám poskytuje univerzální dálnice, NFV jsou různé druhy dopravních prostředků a SDN přidává zkušeného řidiče.

1.2.3 Počátky a vývoj SDN

Určitou inspiraci ve vývoji směrem k SDN lze sledovat v telefonních sítích, kde relativně záhy došlo k oddělení řídicí a datové roviny, k usnadnění správy a možnosti přidávání nových služeb. Ačkoliv je SDN často širokou veřejností chápáno jako synonymum určitých technologií, za SDN se považuje spíše obecný přístup než konkrétní technologie. Vývoj SDN tak zdaleka nebyl a není jenom *Openflow*. Jeho cestu na svět předchází mnoho různých proprietárních i otevřených projektů a pokusů. [9]

Nejčastěji jde o projekty s důrazem na otevřené rozhraní mezi řídicí a datovou rovinou. Z mnohých například rozhraní pro přístup k přepínacím funkcím v *Linuxu* zvané *Netlink* [10], či rozhraní ForCES (*Forwarding and Control Element Separation*) standardizované IETF (*Internet Engineering Task Force*). [11]

Dále jde o projekty s důrazem na centralizované řízení, jako architektura *SoftRouter*, *Routing Control Platform* nebo protokol *Path Computation Element* od IETF. Například *SoftRouter* využíval *ForCES API* pro vzdálené přidávání tabulek pro datovou rovinu přes oddělený kontrolér. [12] *Routing Control Platform* se snažil o okamžitou možnost implementace na současný hardware, a tak pro svojí funkci využíval standartní BGP (*Border Gateway Protokol*). [9]

Proč Openflow?

Openflow je nejznámější protokol používaný pro umožnění komunikace s řídicí vrstvou. Vznik samotného *Openflow* předcházely projekty *SANE* a jeho následovník *Ethane*. Oba vznikaly na akademické půdě Standfordské univerzity v programu *Clean Slate* a na rozdíl od právě vznikajících programů jako *GENI* (*Global Enviroment for Networking*) a evropského *FIRE* se zaměřovaly na menší měřítko sítě, a to sítě v kampusu univerzit. [13]

Práce výzkumníků byla inspirována tzv. *4D* projektem, [14] který navrhuje rozložení řízení sítě do 4 rovin. *Rozhodovací,* která je zodpovědná za vytváření síťové konfigurace. *Propagační*, sbírající informace o stavu sítě a jejich šíření, *objevující*, která je zodpovědná za objevování přímo připojených sousedů a *datová*, pro přepínání síťového provozu.

Projekt *Ethane* nabízel logicky centralizované řešení kontroly přístupu v podnikových sítích. Na základě bezpečnostních pravidel kontrolér rozesílal *„flow“* tabulky daným přepínačům. Mimo jiné se tak tento zjednodušený náhled na přepínač stal základem pro *Openflow* API.

Vývoj SDN se od počátku potýkal s problémem rovnováhy mezi jasnou vizí plně programovatelných sítí a schopností dané řešení reálně uplatnit v praxi. *Openflow* tak pravděpodobně zvítězil díky nalezení rozumné hranice. Částečně limitovanou flexibilitu danou využíváním existujícího hardwaru s univerzálně využitelnými čipovými sadami (*merchant-silicon chipset*) vyvážila možnost rychlého nasazení. Navíc oproti předchozím řešením přinesl mnoho dalších funkcí, které dřívější kontroléry nedokázaly. Rozvoj a rozšiřování nabídky komoditních přepínačů s výše zmíněnými čipovými sadami započal už před vznikem *Openflow*. Díky tomu, že zařízení s těmito čipovými sadami umožňovala například kontrolu přístupu, či monitorování toků, samotná implementace *Openflow* tak spočívala ve snadné aktualizaci firmware. [9]

Podrobnější popis fungování *Openflow* následuje v dalších kapitolách.

Trend otevřeného zdrojového kódu

Vývoj *Openflow*, SDN a obecně virtualizace v sobě nese důležitý milník z pohledu uzavřenosti systémů. Zvyšující se požadavky a tlak zákazníků z různých odvětví jako jsou například provozovatelé sítí, *„cloudů“*, operátoři, finanční společnosti, ale i samotní vývojáři zařízení, podpořili vznik organizací či eko-systémů, které kladou důraz na řešení s otevřeným zdrojovým kódem. Mezi nejznámější patří *Open Network Foundation*, která má na starosti mimo jiné i standardizaci *Openflow*. Dále, jakožto otevřený rámec (*framework*) pro usnadnění implementace SDN a NFV, vznikl *Open Daylight Project*, původně vedený *The Linux Foundation*.

Dalším významným hráčem se stal OCP (*Open Compute Project*). Za jeho počátkem byli inženýři z *Facebooku*, kteří stáli před problémem, jak co nejefektivněji a nejlevněji škálovat firemní infrastrukturu. Projekt se dále rozrůstal a současně nabízí speciálně navržená řešení v podobě serverů, designu racků, napájecích a bateriových modulů. Dále byl vyvinut software pro efektivní správu datových center zvaný FBOSS (*Facebook Open Switching System*) a následně byl zveřejněn jako otevřený zdrojový kód pro podporu vývoje softwarově definovaných datových center. [15] Mezi další důležitý projekt patří ONL (*Open Network Linux*). Jedná se o *Linuxovou* distribuci pro přepínače na základě bílých skříněk (*white box switch*).

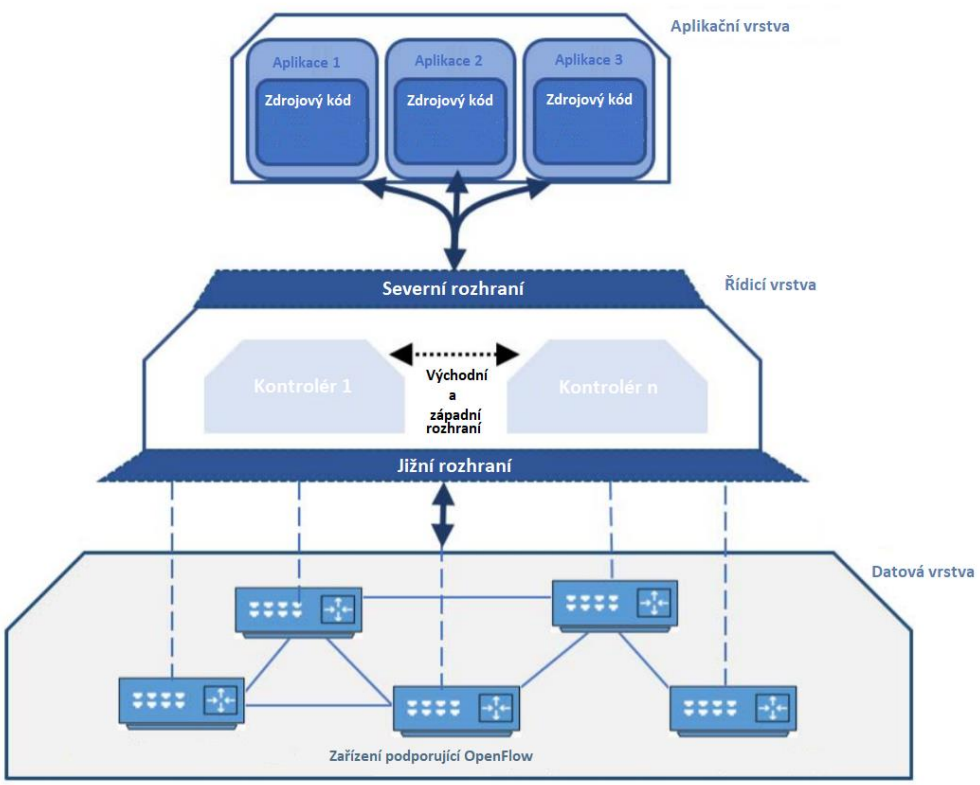
1.3 Architektura SDN

Architekturu SDN popisuje obrázek 1.2. Obecně se jedná o rozdělení směrování do tří rovin, a to aplikační, řídicí a datové. Do datové se řadí veškeré síťové aplikace, které zajišťují požadované funkcionality sítě, přičemž využívají pouze abstraktního pohledu na síť.

Druhá, řídicí vrstva má na starosti řízení a směrování na datové vrstvě. Jedná se o překlad úkolů z aplikační vrstvy na vrstvu datovou pomocí otevřených rozhraní a funguje tak jako brána k přístupu do kontroléru.

Tato vrstva se často skládá z více kontrolérů vhodně rozmístěných v dané síti. Kontroléry pak vystupují jako logicky centralizovaný prvek, běžící jako softwarová platforma NOS (*Network operating system*). Vrstva zpětně poskytuje informace o síťových prvcích aplikacím na aplikační vrstvě. Mezi tyto informace patří například statistiky a události. Rozhraní pro komunikaci mezi aplikační a řídicí vrstvou je nazýváno jako severní rozhraní (*northbound interface*).

Nejnižší vrstva v sobě obsahuje samotnou infrastrukturu a síťové prvky. Jejím hlavním úkolem je směrování paketů na základě instrukcí získaných z řídicí vrstvy. Součástí vrstvy je CDPI (*Control to Data-Plane*) řadič, který se stará o komunikaci s jednotlivými síťovými prvky podporujícími SDN, ať už fyzických či virtualizovaných. Samotné řídicí zprávy a API pro komunikaci je definováno jako tzv. jižní rozhraní (*southbound interface*). [16]



Obrázek 1.2 - Architektura SDN, upraveno z [17]

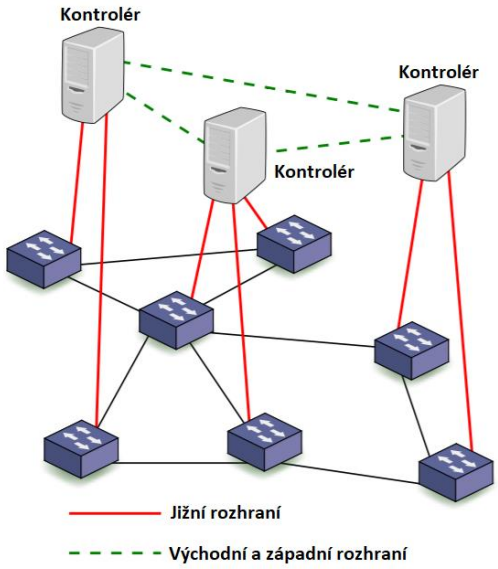
SDN kontrolér či NOS je řídicím prvkem celé sítě. Předkládá celkový přehled o systému a umožňuje centrálně spravovat jemu podřízené síťové prvky. Důležitou úlohou NOS je poskytnutí abstraktního náhledu, řízení služeb a poskytování jednotných API pro vývojáře síťových aplikací. Stejně tak jako u tradičních operačních systémů se vývojáři nemusí starat o konkrétní detaily jednotlivých zařízení.

NOS pro svoji funkci nejčastěji získává informace ze tří zdrojů. Prvními jsou „*flow*“ statistiky generované na směrovačích a pravidelně sbírány kontroléry. Dalšími jsou zprávy vzniklé na základě konkrétních událostí, jako jsou například změny stavu rozhraní, či daného spoje. V případě nové či neznámé „*flow*“ jsou třetím zdrojem dotazy od směrovačů.

Jádro kontroléru či NOS může být charakterizováno jako kombinace dostupných síťových služeb, či různých rozhraní určujících základní funkcionalitu. Mezi nejčastěji využívané patří správa topologie, distribuce konfigurací, sledování stavu a vytížení, posílání notifikací, správa konkrétních zařízení, bezpečnostní mechanismy, hlídání priorit pravidel, řešení jejich konfliktů aj.

Distribuovaná řídicí rovina

Díky možnosti distribuované kontrolní roviny je zaručena větší spolehlivost a lepší škálovatelnost. Prvky na řídicí rovině mezi sebou komunikují pomocí západního a východního rozhraní (*east-bound* a *west-bound*). Mezi jejich funkce patří import a export dat, hlídání konzistence informací, vzájemná upozornění, monitorování a vzájemné předávání podřízených síťových prvků. Velkou výzvou v takto distribuované řídicí rovině zůstává konzistentní a správné rozhodování v celém rozsahu sítě. Dále pak zajištění ochrany proti chybám, kdy v případě pádu či vysoké chybovosti, musí dojít k převzetí povinností některým ze sousedních kontrolérů. S tím je spojený i výběr vhodného umístění a počet kontrolérů v síti.



Obrázek 1.3 - Distribuovaná řídicí rovina SDN, upraveno z [18]

Nejčastějším přístupem je optimalizace na průměrné, či nejhorší možné zpoždění mezi kontrolérem a přepínačem. V případě dotazů na kontrolér dochází ke zpožďování provozu a potažmo celé sítě, protože přepínač čeká na odpověď obsahující pravidla pro daný „*flow*“. V případě omezené šířky pásma dochází ke snížení počtu zpracovaných „*flow*“ v daném časovém intervalu a v případě saturace až ke zvýšení ztrátovosti. [16]

V současné době je vyvíjeno mnoho ať už proprietárních řešení, či těch s otevřeným zdrojovým kódem, které se od sebe navzájem velmi liší v obsažených funkcionalitách, možnostech škálování, modulárnosti či vzájemné spolupráce, hardwarové či čistě softwarové implementaci. Objevují se i platformy pro specifické účely jako jsou například poskytovatelé „cloudových“ služeb, či telefonní společnosti. Nejvýznamnější zástupce možných řešení uvádí kapitola 3.

1.4 Openflow

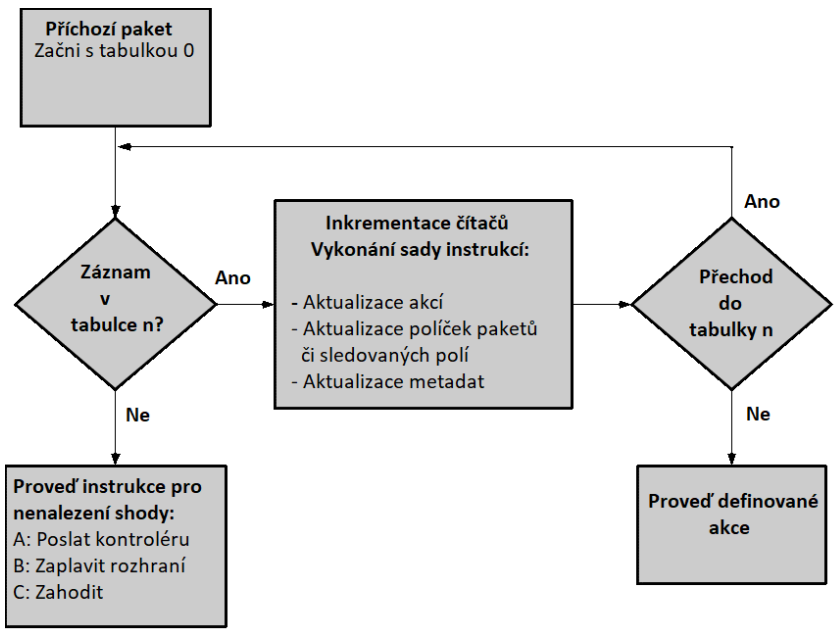
Openflow je jedním z nejrozšířenějších otevřených komunikačních standardů mezi řídicí a datovou vrstvou v SDN. Jeho základ je postaven na technologii Ethernet s využitím definovaných akcí a „*flow*" tabulek. Podkapitola byla zpracována na základě informací ve standardu [19].

Kontrolér spolu s přepínači komunikují pomocí zabezpečeného kanálu, přičemž komunikaci zahajuje přepínač. Pro bezpečný přenos informací využívají TLS spojení na portu 6653. K autentifikaci dochází za pomoci výměny certifikátů a jejich ověření pomocí privátních klíčů. Jakožto alternativa se dle standardu nabízí i spojení pomocí klasického TCP, které však není z bezpečnostních důvodů doporučováno.

Openflow označuje porty na příchozí straně jako vstupní (*inbound)* a na straně odchozí jako (*outbound*). Dále podporuje tři druhy rozhraní: logické, fyzické a rezervované. Logické rozhraní může označovat příslušnost k určité VLAN, tunelu nebo například „*loopback"*. Rezervovaná rozhraní se dělí na povinná a volitelná. Povinná musí podporovat všechna zařízení. Mezi rezervovaná rozhraní patří například možnost zaplavování, či směrování za pomocí klasického přepínání bez *Openflow*.

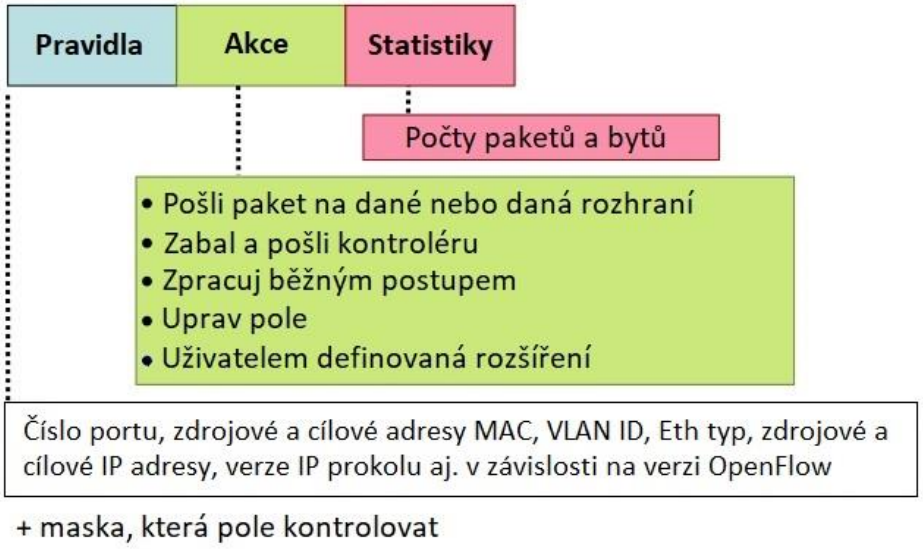
Openflow algoritmus

Po přijetí paketu na vstupním rozhraní, dochází k rozhodování na základě porovnání se záznamy v první „*flow*" tabulce. Pokud nedojde ke shodě, přepínač porovnává hlavičku paketu s dalšími tabulkami, které mají každá své unikátní ID. Tabulek může být až 255. V případě nenalezení shody (*table-miss*) dochází k vyřazení „*flow*", přesměrování na kontrolér, popřípadě k zaplavování více rozhraní. Pokud je nalezena shoda dochází k provedení přiřazených akcí, inkrementaci statistických údajů a směrování „*flow*" na dané výstupní rozhraní.



Obrázek 1.4 - Vývojový diagram protokolu *Openflow,* upraven z [19]

Tabulky *Openflow* se skládají z těchto hlavních částí: pravidla, akce a statistiky. Pravidla se skládají z podmíněných vlastností jako jsou například příchozí port, MAC a IP adresy, čísla TCP portů, příslušnosti k dané VLAN aj. V závislosti na verzi OF se přidávají i položky priorita, další instrukce, „*timeout*“ či „*cookies*“. Priorita pravidel je dána sekvenčně jdoucím číslem tabulky a jejich pořadí v ní. Mezi možné akce patří: směrovat na odchozí rozhraní, směrovat na kontrolér, zahodit paket, zpracovat klasickým směrováním bez OF, či odeslat do zvláštních tabulek.

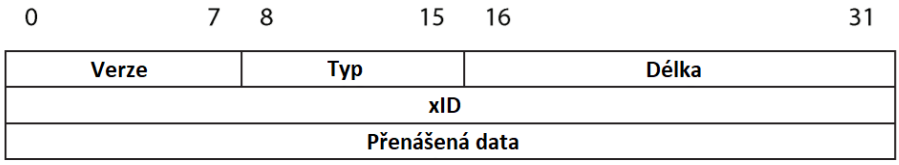


Obrázek 1.5 - *Openflow* tabulka, upravena z [20]

Jednotlivé verze specifikací OF přidávají další podporovaná pole pro rozšíření pravidel jako jsou MPLS, Ethernet, verze IP protokolu, typ transportního protokolu aj. Mezi další přidané prvky patří i tzv. skupinové tabulky, které sjednocují skupinu rozhraní jako jedinou směrovací entitu. Díky tomu lze provést stejné akce na více „flow“, bez nutnosti zvláštní konfigurace pro každou „flow“, nebo například možnost vybírat z množiny portů pro vyvažování zátěže.

Zprávy

Každá OF zpráva začíná jednoduchou hlavičkou, viz obrázek 1.6.



Obrázek 1.6 - Hlavička *Openflow* protokolu

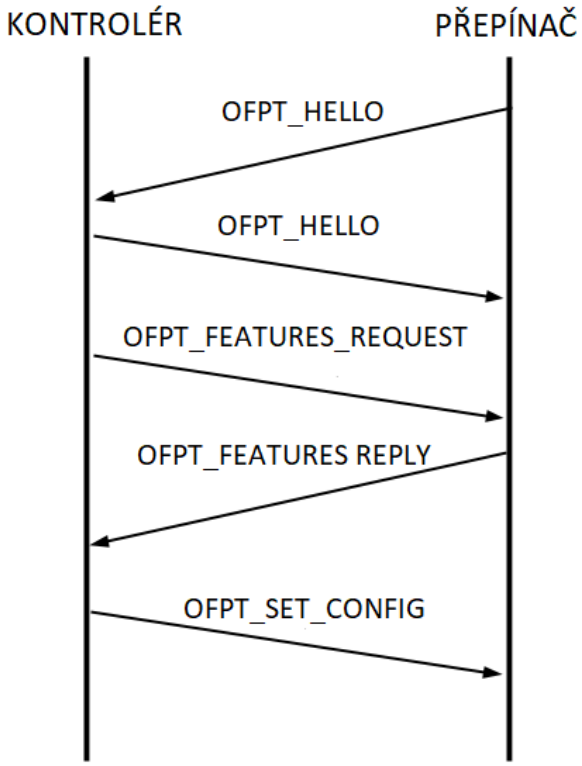
Pole verze označuje verzi použitého protokolu. Hodnota 0x06 označuje současnou verzi *Openflow* 1.5.1. Další pole určuje typ používaných *Openflow* zpráv. Mezi ně patří například *HELLO, ERROR, ECHO_REQUEST, FEATURES_REQUEST* aj. Celkovou délku zprávy v bytech určuje pole délka, xID je označení transakce ke které daný paket spadá. Během odpovědi je použito stejné xID.

Obecně jsou v OF používány tři skupiny zpráv. Mezi kontrolérem a přepínačem, asynchronní a symetrické. Zprávy mezi kontrolérem a přepínačem jsou zahajovány kontrolérem a patří mezi ně například dotazy na podporované funkce nebo aktualizace konfigurací směrovačů.

Jako symetrické zprávy jsou označovány odpovědi směrovače, informace o chybách, stavová hlášení, *HELLO* z úvodního *„handshake“* nebo *ECHO* sloužící jako *„keep-alive“* zpráva. Mezi asynchronní zprávy patří ty zasílané bez vyžádání kontroléru, například přesměrování na kontrolér, informace o odstranění záznamu o „flow“ z tabulky aj.

Sestavení spojení s kontrolérem

Před sestavením spojení probíhá výměna zpráv *HELLO* z obou komunikujících stran. Paket v sobě obsahuje nejvyšší podporovanou verzi OF. Pokud se zařízení neshodnou na používané verzi, příjemce musí odeslat zprávu *HELLO_FAILED* se znakem *INCOMPATIBLE*. Po sestavení spojení probíhá *„handshake“*, kdy kontrolér vyžádá zprávou *FEATURES_REQUEST* podporované funkce a ID přepínače. Přepínač poté odpovídá zprávou FEATURES_REPLY.



Obrázek 1.7 - *Openflow „handshake“*

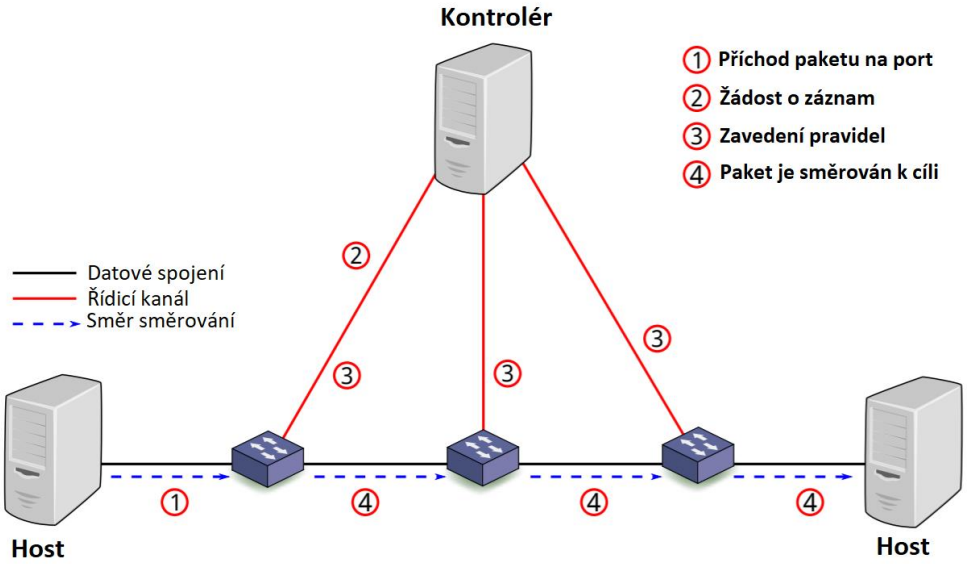
V případě ztráty spojení přechází přepínač do *„fail stand alone“* nebo *„fail secure“* módu. Ve *„fail stand alone“* módu se pakety směrují pomocí vyhrazeného rozhraní *NORMAL*, kdy se zařízení chová jako klasický směrovač bez OF. V druhém modu pak směrovač zahazuje veškerý provoz až do obnovení spojení.

Proaktivní a reaktivní „*flow*“

V *Openflow* dochází k porovnávání políček se záznamy v pravidlech se specifickými políčky v hlavičkách paketů. Tyto záznamy jsou pro rychlé vyhledávání ukládány v obsahem adresované paměti (TCAM). Pro velmi vysokou cenu těchto pamětí, je jejich kapacita v síťovém prvku omezená, díky čemuž se do ní vejde jen relativně malé množství záznamů. Existují tak dvě možnosti, jak spravovat záznamy ve „*flow*“ tabulkách.

Proaktivní přístup drží všechny záznamy trvale v paměti před tím, než je bude potřebovat použít. Tento přístup je náročnější na správu hierarchie paměti, přičemž tolik nezatěžuje kontrolér a je tak více robustní proti chybám a případným přetížením na řídicí rovině. Nevýhodou je pak značná rozsáhlost uložených informací. V případě sítí provozovatelů a využití BGP protokolu se může jednat o tisíce IP záznamů.

Za reaktivní přístup je považováno zavádění záznamů až v případě potřeby. Přepínač se tak v případě, kdy nemá pravidlo pro příslušné „*flow*“ ve své tabulce, dotazuje kontroléru. Tento přístup je umožněn díky časovači, kdy v případě nevyužití záznamu je po uplynutí této doby vymazán z paměti. Může tak pro svoji práci využít výrazně menší paměť, ale v závislosti na vytížení kontroléru dochází ke zpoždění na jednotlivých přepínačích. Obě varianty se vzájemně nevylučují a je tak možné oba postupy vzájemně kombinovat.



Obrázek 1.8 - Získávání záznamů při reaktivním řízení „*flow*“, upraveno z [18]

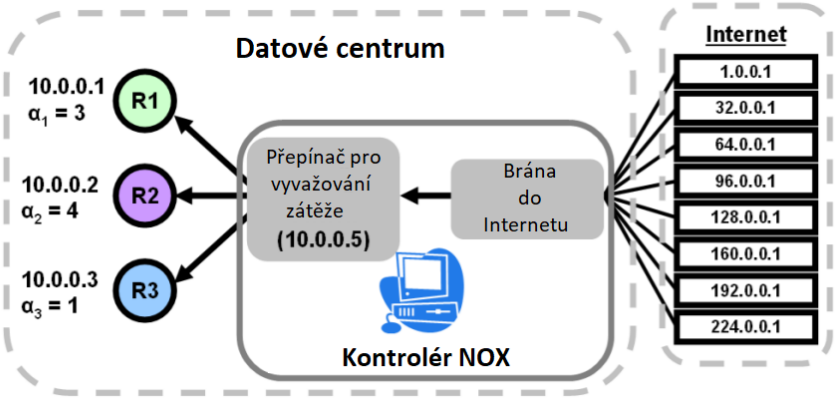
2 Současné možnosti využití SDN a NFV

Nasazení SDN má obecně největší přínos v řízení síťového provozu a ve sjednocení jeho správy. Tato kapitola uvádí souhrn současných projektů, které dokládají rozvoj konceptu SDN a NFV. Mezi nejvýznamnější funkcionality patří vybírání optimální cesty, maximalizování využití agregace cest, provádění vyvažování zátěže, řízení přístupu, alokace šířky pásma, dodržování QoS či SLA, efektivnější využívání prostředků, šetření elektrické energie, umožnění celkové virtualizace, otázky bezpečnosti aj.

2.1 Vyvažování zátěže

Vyvažování zátěže je jedna z nejstarších aplikací uvažovaných pro SDN, která je v klasických sítích často řešena pomocí dedikovaného zařízení. Pokud bychom uvažovali o nejjednodušším řešení, ve kterém bychom každému uživateli přiřadili záznam do *„flow“* tabulky, dojdeme brzy k závažnému nedostatku, kterým je špatná škálovatelnost.

Nabízí se tak efektivnější řešení uvedené například v [21]. Jedná se o tzv. proaktivní vyvažování zátěže za pomocí *„wildcard“* pravidel. *Openflow* přepínač automaticky distribuuje provoz přímo k serverům, na kterých běží repliky dané služby. Kontrolér monitoruje provoz a na základě citlivě zvolené prahové hodnoty počtu toků je rozkládá tak, aby nedošlo k vytvoření úzkého hrdla. Dochází také k agregaci dotazů od uživatelů na základě rozsahů IP, díky čemuž může směrovat velké skupiny uživatelů a snižuje se tak množství dotazů na kontrolér. Dosažené výsledky ukazují efektivní řešení bez nutnosti dedikovaného zařízení. Díky využití SDN může přidání nového serveru do sítě proběhnout dynamicky a flexibilně dle aktuálního vytížení výpočetních prostředků a daného segmentu sítě.



Obrázek 2.1 - Základní model vyvažování zátěže z pohledu přepínače, upravený z [21]

2.2 Dynamické řízení přístupu

Ve spojení s řízením přístupu se často využívá standard IEEE 802.1X. Jako příklady využití v sítích SDN uvádíme následující dva projekty.

flowNAC

flowNAC je podle [22] NAC (*Network Access Control*) založený na práci s „*flow*" a rozpoznávání jednotlivých služeb na základě požadavků uživatele. Každá služba je zde jednoznačně definována jako skupina určitých „*flow*", proto může být jedním uživatelem nezávisle vyžádáno a zároveň najednou autorizováno více služeb. Další služby mohou být dle požadavků v průběhu zpřístupněny či zablokovány. SDN tak přináší možnost pracovat s NAC detailněji a flexibilněji a to na základě konkrétně vyžádaných služeb.

Pro autentifikaci uživatelů využívá modifikovanou verzi IEEE 802.1X bez nutnosti portálu zadržení (*captive portal*) a na základě proaktivního „*flow*". Oproti klasické architektuře NAC, založené na kontrole přístupu přímo nad fyzickými rozhraními zařízení (PNAC), *flowNAC* rozkládá proces kontroly přístupu a vykonávání pravidel. Kontrola přístupu v PAE *(port access entity)* je přesunuta do centralizovaného SDN kontroléru a vykonávání pravidel obstarává PAC *(port access controller),* který je ponechán na přepínačích.

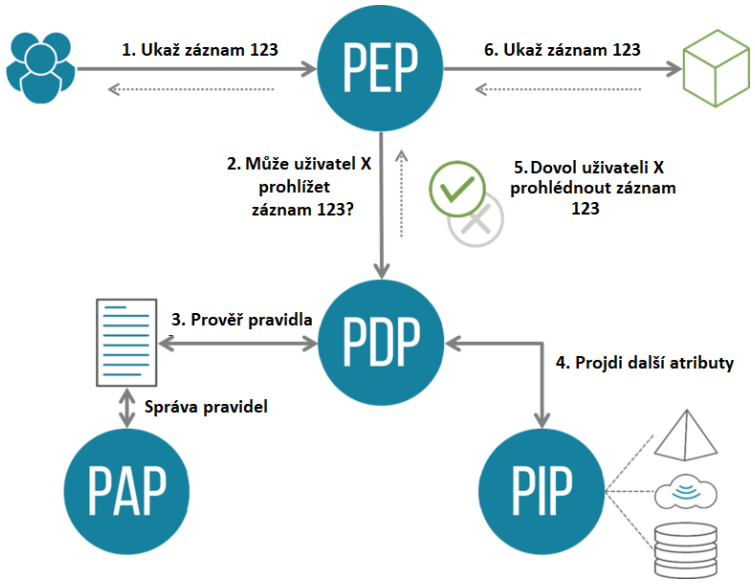
V případě využívání stejného fyzického rozhraní více uživateli, dochází u klasického PNAC k vytvoření více virtuálních rozhraní, přičemž každé virtuální rozhraní je identifikováno MAC adresou uživatele. Klasický PNAC však nerozlišuje jednotlivé služby a v případě porušení pravidel dochází ke kompletnímu zamezení přístupu k síti.

Hlavní výhodou oproti klasickému řešení nad rozhraními je tak rozšíření funkcionality a možnosti většího rozlišení na základě požadavků na konkrétní služby. Nevýhodou je větší náročnost na to, aby poskytovatelé správně a jednoznačně nadefinovali služby, jinak by mohlo docházet ke špatné identifikaci nebo kolizím.

Flow Identity

Jedná se o virtualizovaný NAC využívající *Openflow* protokol. *Flow Identity* implementuje rámec 802.1X kombinovaný s autorizací skrz stavový (*stateful*) *„firewall"*. Pravidla v RBAC (*role-based access control*) jsou definována na základě rolí koncových bodů či uživatelů (například příslušnost k určitému oddělení), nebo na základě prokazatelné identity. [23]

PEP (*Policy Enforcement Point*) přijímá a zpracovává požadavky od uživatelů a dotazuje se PDP (*Policy Desicion Point*), který zhodnotí, zdali má uživatel právo přístupu. PIP (*Policy Information Point)* udržuje identitu uživatelů.



Obrázek 2.2 - Schéma řízení přístupu na základě rolí, upraveno z [24]

Každý požadavek na službu, potažmo *„flow“*, je před vpuštěním porovnáván s tabulkou *„firewall“* pravidel. *FlowIdentity* využívá stejného principu oddělení kontroly přístupu od vykonávání pravidel jako *flowNAC* a zároveň upravuje rovinu vykonávání pravidel. Pravidla mohou být aktualizována dynamicky a s okamžitou platností. Stavové vyhodnocování přináší větší bezpečnost, ale může přinášet problémy se škálovatelností a výkonem.

2.3 Kvalita služeb (Quality of Service)

SDN je hojně využíváno pro zaručení dané třídy QoS, tj. schopnost sítě rozlišovat provoz a upravovat parametry na základě stanovených priorit. K měření parametrů sítě dochází aktivně nebo pasivně. Obecně je pasivní měření méně přesné, a ne vždy proveditelné, ale nepřináší další režii. Aktivní měření přidává do sítě další provoz v podobě měřících paketů, ale vykazuje přesnější výsledky parametrů jako fázové chvění nebo zpoždění. Z mnoha zveřejněných projektů uvádíme tyto následující.

Rámec pro řešení QoS

Navrhovaný rámec se podle [17] skládá z několika logických bloků. Tyto externí bloky vznikly jako rozšíření směrovacích funkcí kontroléru. Výhodou dělení na samostatné bloky je možnost každou funkcionalitu implementovat odděleně a moci tak měnit konkrétní řešení či využívané parametry, podobně jako například u vrstvového modelu sítě. Mezi použité bloky patří: monitorování, vybírání cesty, blok přípravy pravidel a konfigurace.

Monitorování sleduje současné využití rozhraní, měří zpoždění spojů aj. Modul vybírající cestu se opírá o neváženy *Dijkstra Shortest-path* algoritmus nebo *QoS-enabled route selection*, přičemž může být nahrazen jiným, komplexnějším algoritmem. Při výběru cesty bere toto řešení jako metriku nejkratší cestu, která má současně nejmenší zpoždění. Pro funkcionalitu monitorování sleduje současné vytížení daného rozhraní. Rámec umožňuje rozšíření o další metriky, jako jsou fázové chvění, aktuální množství packetů, ztrátovost a další.

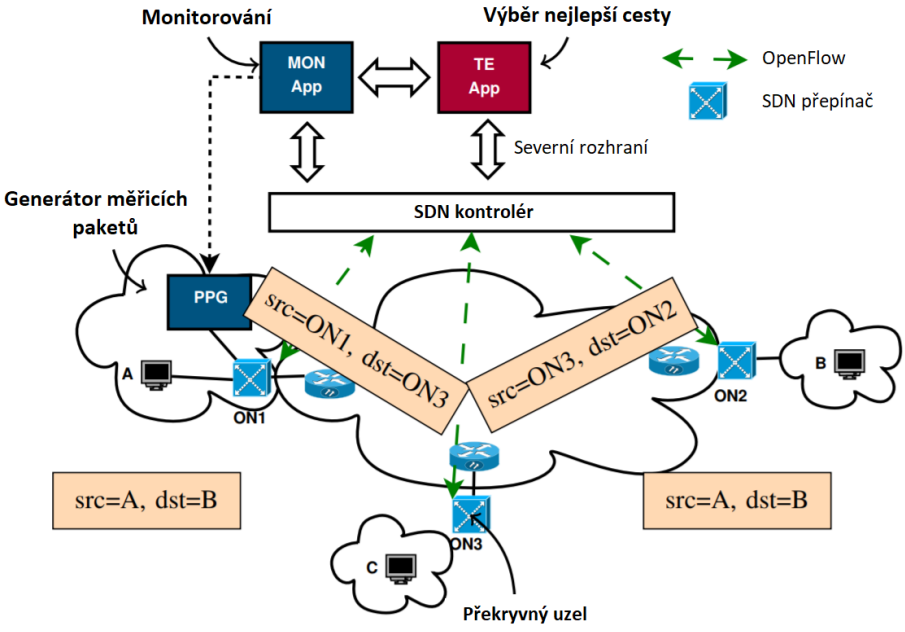
Výpočet cest je současně jednoduše realizován na základě grafových informací z kontroléru. Cesty se stejným počtem skoků jsou tak porovnávány na základě zadaných metrik. Vybraná cesta je pak porovnávána s výsledkem algoritmu. Nakonec je vybraná cesta předána modulu přípravy pravidel, kde se specifikují konkrétní časovače a priority pro dané uzly nebo aplikace. Odtud jej dále předává kontroléru pro rozeslání.

Navrhovaný rámec byl testován za použití *OpenDayLight* kontroléru na topologii *„fat-tree“*, která je velmi často využívána v datových centrech. Autoři uvádí, že na základě experimentů došli v porovnání jejich řešení a klasické SDN architektury k prokazatelným výsledkům. V jejich měření došlo ke snížení zpoždění až o 57 %, snížení fázového chvění o 25 % a ztráty paketů až o 67 %.

QoS překryvná síť

Tato práce [25] popisuje možnost využití překryvných sítí pro snadno škálovatelné řešení QoS. Díky využití SDN a konkrétně *Openflow* protokolu umožňuje na každém přepínači upravovat hlavičky paketů na základě daného *„flow“* a podle vybrané cesty.

Architektura zobrazená na obrázku 2.3 je virtuálně centralizovaná a skládá se z překryvné sítě připojené pomocí překryvných uzlů. To jsou přepínače podporující *Openflow*, spojené přes Internet. Dále se skládá z těchto hlavních bloků: *„TE app“* – vybírá nejlepší trasu na základě vybraných metrik, *„MON app“* – poskytuje *„TE app“* zpětnou vazbu o stavech sítě, dále generátoru paketů pro měření a samotného kontroléru.



Obrázek 2.3 - Architektura *QoS-aware overlay network*, upraveno z [25]

Vzhledem k možným vzdálenostem mezi přepínači bylo zvoleno lokální aktivní monitorování, založené na umístění paketového generátoru na každém překryvném uzlu, které jsou centrálně řízeny přes *„MON app“*.

„TE app“ musí po přijetí nového paketu rozhodnout, kterou cestou daný paket a potažmo celý *„flow“* bude dále směrován. V návaznosti na to pak kontrolér určuje danému *„flow“* identifikátor a dále jak s ním nakládat na základě daného stavu cest a žádaného QoS. Mezi nejčastější operace patří změna IP adres. Kontrolér tedy nakonfiguruje dané přepínače a ty pak předávají pakety na základě těchto pravidel. Toto řešení tak připomíná NAT/PAT (*Network Address/Port Address Translation*), přičemž ale zůstává transparentní pro koncové uživatele, neboť hodnoty jsou před dosažením samotného koncového zařízení navráceny k původním.

2.4 SD-WAN

SD-WAN (*Wide Area Network*) přináší softwarově definovaný přístup ke směrování do WAN sítí. Funguje tak mezi sítěmi či datovými centry v různých, často velmi vzdálených regionech.
Můžeme jej tak brát jako aplikaci SDN, která se primárně zaměřuje na sítě WAN či na jádro sítě daného poskytovatele služeb.

Jedná se o další vývojový krok z tzv. hybridních WAN sítí. Ty jsou obecně definovány jako WAN sítě, které se skládají z MPLS a jednoho čí více redundantních spojů. Využívá se tak výhod daných MPLS jako je spolehlivost, dané QoS, či možnost garantovat určitá SLA (*Service-level Agreement*). [26] Takovýto spoj je pak využíván pro kritické služby společnosti, kdežto přes zmíněné další spoje prochází provoz k Internetu a fungují také jako záloha pro MPLS. Mezi nevýhody tohoto řešení patří pád relace při náhlé nedostupnosti a neefektivní hospodaření s dostupnou šířkou pásma.

SD-WAN tak využívá stále více se vyskytující hybridní architektury a přidává výhody samotného SDN jako je kontrolér, který je schopný realizovat pravidla v závislosti na právě využívaných aplikacích. Dále přináší lepší přehled o síti a centrální správu. Díky tomu umožňuje změny cest dynamicky a na základě kontextu daného provozu.

Dochází ke zvýšení bezpečnosti a to nejen díky odstínění od použité transportní technologie. Pokud se chce účastník na pobočce firmy, bez využití SD-WAN bezpečně připojit například k veřejnému *„cloudu“*, prochází často spojení přes centrální bezpečnostní bod v korporátním datovém centru, což přináší další zpoždění a nadužívání nákladných MPLS linek. Oproti tomu se uživatel za pomoci SD-WAN může bezpečně připojit napřímo. (Za předpokladu využití *„firewallu“* schopného pracovat na vrstvách 4 až 7). Tato technologie totiž umožňuje přímé rozdělování provozu na základě použité aplikace do daných VN a VPC (*Virtual Private Cloud*), což přispívá ke vzniku bezpečné konektivity. [27]

Jedním z důležitých faktorů pro změnu bylo i relativně obtížné a zdlouhavé přidávání nových MPLS spojů. Kdežto při využití klasických služeb poskytovatelů spojení se připojení nové pobočky může provést v řádu několika hodin. V neposlední řadě toto řešení nabízí snížení výdajů za provozování MPLS spojů.

2.5 Šířka pásma a řízení přetížení

Rezervování šířky pásma

Rezervování šířky pásma *(Bandwidth calendaring)* je technika umožňující dočasně posunout budoucí požadavky na šířku pásma, což umožňuje efektivnější využití síťových kapacit. Díky tomuto posunu mohou sítě nabídnout relativně levnou konektivitu s garancí QoS a šířky pásma zejména pro velké bloky dat zpracovávané například ve velkých datech.

Není reálné předpokládat dokonalou znalost budoucích požadavků a tak oproti řešení tzv. „*offline*" verzí kalendáře, kde operátor přesně zná budoucí rezervace, se autoři [28] snaží o implementaci online verze, u které je potřeba rychlého rozhodování a následného informování žádajících aplikací. Největší výzva vychází ze samotné povahy problému, kdy dochází k neustálým změnám parametrů a příchodům nových „*flow*". Operátoři tak nemohou přesně předpovídat parametry jako jsou počet, čas příchodu, trvání či objem daných „*flow*".

Navrhovaný systém umožňuje tvorbu online kalendáře bez úplné znalosti budoucích nároků na šířku pásma, což představuje jakýsi kompromis mezi *„offline"* řešením a složitým heuristickým odhadem budoucích požadavků. Zpráva popisuje 2 algoritmy (*OneShot* a *Postpone*), které rozhodují o přijetí, plánování a směrování daného provozu. Hlavní snaha je zaměřena na přijetí co největšího objemu dat v čase. Práce dále připouští možnost odmítnutí menších toků kvůli malému vytížení kapacity a současnému zabrání přenosového média.

Rozhodování o výběru cesty probíhá na základě funkce zvané *„admission price"*, která je definována tak, že její hodnota exponenciálně roste s vytížením daného spoje, což vede k menší pravděpodobnosti jejího zvolení. Je dáno časové rozhodovací okno, rozdělené na sadu stejně trvajících intervalů - tzv. epochy.

Algoritmus *OneShot* vybírá pro daný požadavek ze všech možných epoch v současném okně tu, která vygeneruje cestu s nejmenší *„admission price"*. Obecně předpokládá nižší vytížení v budoucnu, vybírá tak z posledních epoch daného okna a tím zpožďuje daný požadavek. Oproti tomu se *„Postpone"* snaží vybírat cestu v prvních epochách, kde zisk převáží *„admission price"* a snaží se tak zabírat co nejmenší část okna, přičemž ale nerozhoduje tak rychle jako *OneShot*. Oproti přístupu, který nebere v potaz *„admission price"* a rozhoduje se pouze na základě nejmenší váhy cesty, byla změřena až o 20% větší propustnost na reálném provozu.

Dynamické řízení šířky pásma (v PON)

Článek [29] pojednává o možnosti využití SDN v řešení problému sdíleného média v PON (*passive optical network*). Konkrétně jde o návrh nového rámce, přinášejícího řešení dynamické alokace šířky pásma ve směru *„upstream"*, který je najednou sdílený více uživateli. Kontrolér tak umožňuje centrálně spravovat pravidla pro alokaci šířky pásma různého provozu od různých ONU (*Optical Network Unit*) jednotek. Navržená architektura obsahuje modul pro sledování provozu, modul pro správu vlnových délek a modul pro správu šířky pásma.

Autoři navrhují rozdělení provozu do tří prioritních skupin, a to *„expedited forwarding“, „assured forwarding“* a *„best effort“*. Po příchodu dat od uživatele na ONU jsou zařazena ve frontě podle dané skupiny a je zaznamenán čas příchodu prvního paketu. Dále jsou podle definovaných *„flow“* pravidel a navrhovaného algoritmu autorizovány jednotlivé požadavky na alokaci šířky pásma. Na základě statistické analýzy požadavků kontrolér neustále upravuje záznamy ve *„flow“* tabulkách. Následná simulace ukazuje, že je možné technologii PON obohatit o funkci QoS a také fakt, že zpoždění bylo zmenšeno až o 23 %.

Řízení přetížení

Řízení přetížení (*Congestion control*) popisuje postup proti přetížení sítě, či její části. To nejčastěji nastává v situacích, kdy se na určitém místě zpracovává velké množství paketů a z toho vycházející nedostatek systémových prostředků zařízení, či kapacity sítě. Tak dochází k poklesu kvalitativních parametrů celé sítě, jako jsou například propustnost, zpoždění či fázové chvění.

Mechanismus použitý v [30] se snaží o nalezení alternativních tras stejné délky a rozložení zátěže namísto zatěžování jednotlivých linek v topologii. Použitou topologií je *„fat-tree“*, která se vyznačuje stejným počtem spojů mezi sousedy na stejné hladině jako mezi jejich rodiči. Čím blíže se tak nacházíme ke kořeni, tím více cest je mezi danými uzly.

Algoritmus je popsaný v základních šesti krocích. Pomocí *REST API* jsou získávány informace o topologii a stavu zařízení. Ve druhém kroku jsou vybrány dva uzly mezi kterými se má kontrola provádět a je použit *Dijkstrův* algoritmus pro nalezení možných nejkratších cest mezi nimi. Dále se určuje cena vybraných spojů pomocí údajů z použitého *OpenDayLight* kontroléru. Ve čtvrtém kroku se vybere cesta s nejnižší cenou, tj. nejnižší vytížení a následuje zadání pravidel a jejich distribuce. Pro přidání dynamiky se celý průběh opakuje každou minutu.

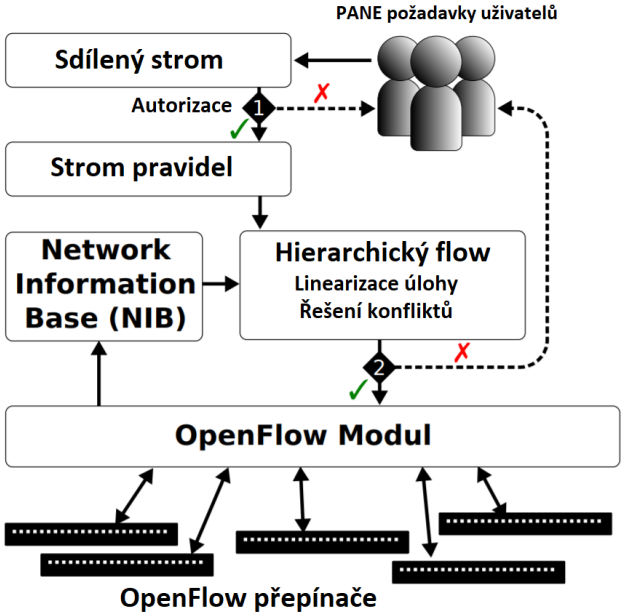
2.6 SDN a síťová virtualizace

K realizaci síťové virtualizace (*network virtualization*) jsou využívány explicitní záznamy ve „flow“ tabulkách. Práce [31] přináší tzv. hierarchické *„flow“* tabulky (HFT). Hierarchická pravidla jsou často využívána v místech, kde dochází ke sdílení zdrojů mezi více entitami. Tato pravidla umožňují snadné předávání pravomocí a řešení konfliktů. Výše zmíněné tabulky tak udávají sémantiku pro rozhodovací stromy pravidel, které jsou dále využívány při směrování v SDN.

Pravidla jsou organizována jako stromy, kde každá komponenta může nezávisle rozhodnout o akci, která se provede nad daným paketem. Jakmile dojde ke konfliktu v rozhodování těchto nezávislých subjektů, HFT jej vyřeší za pomoci uživatelsky definovatelných operátorů, které se vyskytují u každého uzlu daného stromu. Zde se jedná o dva základní operátory, které je možné libovolně upravovat a přidávat. První z nich je „dítě přepisuje rodiče“ a druhé „explicitní zákaz přepisuje povolení“. Autoři dále popisují vytvoření překladače (*compiler*) pro přidání této funkcionality na přepínače podporující *Openflow*.

HFT jsou autory využívány v tzv. participativních sítích *(participatory networking)*. [32] To obnáší zpřístupnění API mezi koncovými uživateli a řídící rovinou. Uživatelé tak mohou číst současný stav sítě a dále upravovat parametry, či měnit požadavky pro zasílaný provoz přímo řídicí rovině.

Vzniká tak kontrolér zvaný *PANE*, který předává uživateli právo na čtení a zápis s možností omezení ze strany síťového administrátora. *PANE* řeší základní otázky: jak bezpečně zpřístupnit tyto možnosti uživateli, jak přidělovat pravidla a jak řešit konflikty mezi požadavky uživatelů. Tato platforma přináší další krok směrem k síti jako službě (*network as a service)*, kdy síť mohou uživatelé, či jejich aplikace kontrolovat dle svých potřeb.



Obrázek 2.4 - Architektura systému *PANE,* upravená z [32]

2.7 NFV a řetězení funkcí

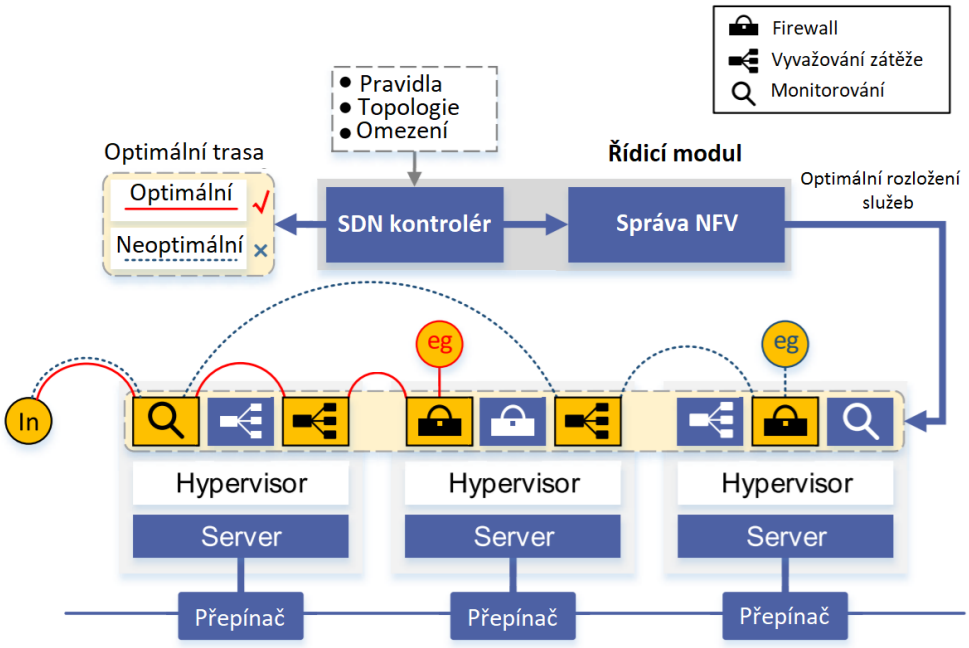
Tato podkapitola byla zpracována na základě informací obsažených v průzkumu [33]. NFV umožňuje transformaci síťových služeb z dedikovaného HW do čistě softwarové báze jako virtuální stroje na obecných bílých skříňkách s kontrolou „*hypervisoru*“ nebo jako služby na virtualizovaných mezilehlých zařízeních (*middlebox*). To přináší významné snížení pořizovacích nákladů a díky SDN můžeme snáze a efektivněji přivést na takto implementované služby provoz.

V současných sítích je velké množství možných funkcí realizováno jako dedikované zařízení, přičemž by mohly být realizovány pomocí NFV. Mezi často používané patří NAT, „*Firewall*“, IDS, vyvažování zátěže, WAN optimalizace, prvky v mobilních sítích jako například *Home Location Register*, *Packet Data Network Gateway*, *NodeB* a mnoho dalších. V případě příchodu nové služby musí být nainstalován a zapojen nový HW, což stojí čas, peníze a v neposlední řadě zvyšuje komplexnost sítě.

Řetězení

Propojování instancí daných služeb, často označované jako řetězení, je umožněno postupným mapování příchozích paketů na řetězce služeb a efektivním směrováním mezi jednotlivými instancemi.

Obrázek 2.5 slouží jako příklad řetězení služeb. Jako vstup bere uživatelem definovaná pravidla. Řídicí modul přiřazuje jednotlivé služby a kontroluje jejich optimální funkci. SDN kontrolér se stará o efektivní směrování, distribuci pravidel, či sledování aktuálního vytížení. Funkce jsou tak řetězeny centralizovaným kontrolérem a provoz je směrován do vytvořených řetězců. Mezi současné výzvy v implementaci patří optimální alokace zdrojů, flexibilita uspořádání služeb v řetězci, umístění v síti či plnění daného QoS.



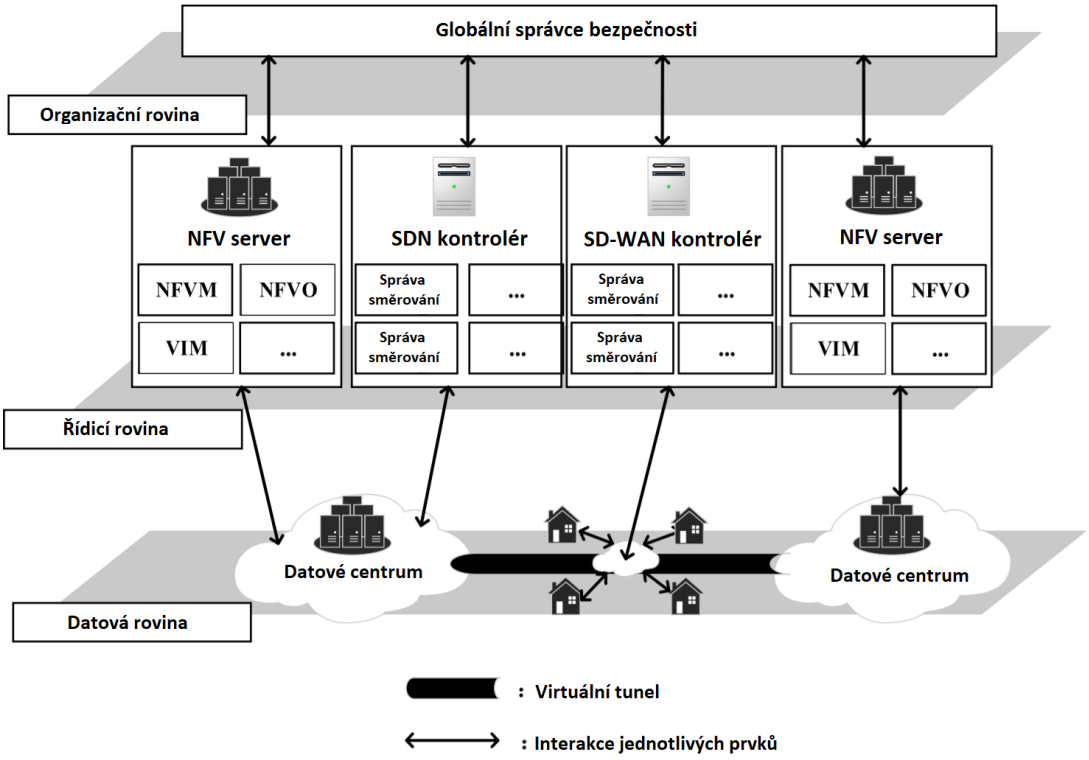
Obrázek 2.5 - Řetězení funkcí v SDN/NFV architektuře, upravené z [33]

Řetězení bezpečnostních služeb

Určení skladby služeb v řetězci je obecně komplexní problém. V případě SSC (*Security Service Chaining*) je důležitý kompromis mezi porušením konzistence a třídy bezpečnosti a požadavky na výpočetní výkon a jeho optimální využití. Obecně je tento problém klasifikován jako *NP-hard*.

Autoři práce [34] se snaží o řešení pomocí heuristického algoritmu *breadth first search*, který slibuje řešení velmi blízké optimálnímu v polynomiálním čase. Obrázek 2.6 popisuje využívanou architekturu založenou na SDN a NFV. Díky tomu dochází ke zkrácení vývoje, snížení nákladů na správu a umožňuje rychlé nasazení nových aplikací. V neposlední řadě mohou být jednotlivé služby libovolně rozprostřeny po síti a dynamicky upravovány na základě požadavků zákazníka.

Model rozlišuje tři základní roviny. Na *organizační rovině* je umístěn *globální správce bezpečnosti*, který přijímá a analyzuje bezpečnostní požadavky od uživatelů. Ty zpracovává a získává z nich například počty či typy instancí. Dále vybírá a efektivně organizuje instance na základě výsledků autory navržené optimalizační úlohy. *Řídící rovina* se skládá z NFV serveru, který má na starosti zřizování instancí služeb a jejich vhodné umísťování, o čemž informuje SDN kontrolér. Ten vybírá cesty, které jsou dále ve formě „*flow*" tabulek distribuovány a používány pro řízení „*flow*" skrz dané instance.



Obrázek 2.6 - Popis základních složek řetězení funkcí, upraveno z [34]

2.8 Bezpečnost

Ochrana proti DNS zesilujícímu útoku

Navrhované opatření [35] proti tomuto typu útoků (*DNS amplification attack*) za pomocí DNS serverů se skládá z několika fází. V první fázi dochází k detekování útoku a co nejrychlejší obraně oběti. Dále se snaží vyselektovat využité jednotlivce z *„botnetu"* a izolovat je ze sítě. Na základě měření je autory toto řešení považováno za málo náročné na využitý výpočetní výkon, výrazně neomezuje normální DNS požadavky a urychluje schopnost sítě obnovit svoji funkčnost.

DNS amplification je útok využívající DNS *ANY* paket, který je zasílán velkým množstvím hostů často spojených v *„botnet"*. V zasílaných výzvách je falšovaná zdrojová adresa, která je zaměněna za adresu oběti. Potom, co DNS server zpracuje požadavek, odesílá blok záznamů v relativně velkém množství paketů a oběť je tak zahlcena DNS odpověďmi.

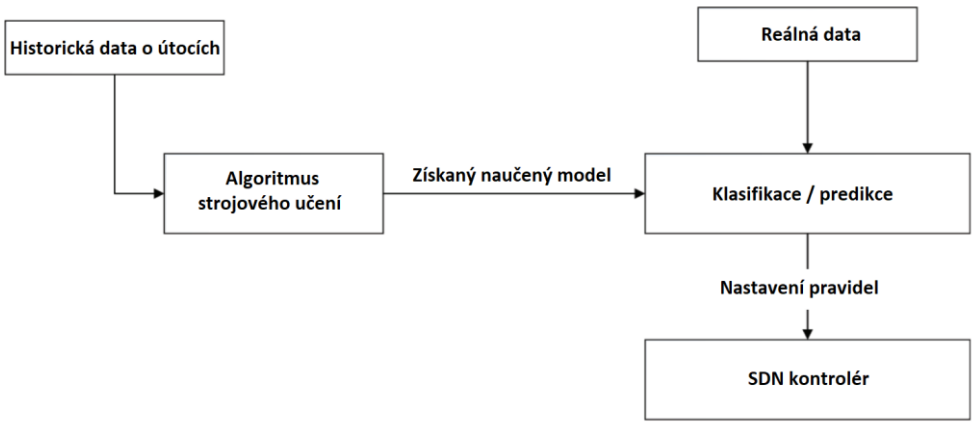
Mechanismus obrany pracuje za pomoci SDN architektury. V normálním stavu všechny SDN přepínače monitorují počty paketů požadující služby DNS. Pokud dojde k překročení hranice *V*, informují o situaci kontrolér. Po *N* událostech od daného přepínače, kontrolér spočítá entropii hodnot zdrojových adres. Pokud je entropie menší než nastavitelná hranice *H*, dochází k přechodu do druhé fáze, kde se zkoumají DNS pakety z podezřelého portu. Pokud se liší počty odpovědí od počtu požadavků o *L*, jedná se pravděpodobně o útok, dojde k zahazování paketů a vstupu do fáze selekce a izolace hostů v „*botnetu*".

Použití strojového učení při odhalování útoků

Autoři [36] poukazují na možnost využít strojového učení pro identifikaci potenciálně nebezpečných připojení a cílů. Umožňuje tak vytipování nejzranitelnějších hostů v SDN, na které by mohl být nejpravděpodobněji veden útok. Díky tomu může kontrolér autonomně reagovat a efektivně upravovat pravidla i pro více souběžně probíhajících útoků.

Díky strojovému učení je možné uživatele klasifikovat na základě jejich specifických charakteristik a chování, na běžné uživatele a potenciální útočníky. Útočníci se vyznačují určitými vzorci, jako je například koordinovanost útoků, sdílení slovníků s hesly aj. Algoritmus tak předvídá část sítě, kde by mohlo k útoku dojít a za pomoci SDN blokuje přístup podezřelých uživatelů. Dochází k blokaci celého adresního rozsahu dané podsítě, neboť se ukazuje, že útočníci většinou využívají z rozsahu více adres. Nedostatkem tohoto přístupu je možnost dočasného omezení konektivity běžných uživatelů.

Práce popisuje implementaci 4 algoritmů a jejich hodnocení v závislosti na efektivitě a přesnosti určení provozu. Mezi testované algoritmy patří: *C4.5, Bayesian Network (BayesNet), Decision Table (DT)* a *Naive-Bayes.* Obecnou architekturu popisuje obrázek 2.7.



Obrázek 2.7 – Algoritmus strojového učení v SDN, upraveno z [36]

S využitím historických dat dochází k trénování modelu. Model dále za zpracovávání současných dat předvídá potenciální útoky a instruuje SDN kontrolér k rozesílání pravidel na dotčené síťové prvky. Přesnost určení závisí mimo odlišnosti provozu také na upravitelném parametru α, který slouží jako prahová hodnota pro podezřelý provoz. Experiment ukázal, že průměrná přesnost predikce je při využití *Bayesian Network* algoritmu necelých 92 %.

DDoS

Útoky typu „*DDoS*" se mimo větší rozměry a častější výskyt stávají také více sofistikované, protože se zaměřují na konkrétní služby v síti a jsou tak méně nápadné a náročné na šířku pásma. Díky tomu jsou hůře odhalitelné a více efektivní, neboť míří na dříve odhalené slabiny v konkrétních službách jako jsou například VoIP, DNS, či HTTP. [37]

Autoři dále rozdělují mechanismy proti „*DDoS*" útokům na základě jeho umístění:

1. Mechanismy na základě zdroje (*source-based*) jsou umístěny blízko k zákazníkovi, kde se předpokládá možný vznik útoku z jeho sítě. Nejčastěji se jedná o detekci anomálií v provozu, či vstupně/výstupní (*ingress/egress*) filtrování, které vybírá podezřelé pakety na základě validace zdrojových IP adres.

2. Mechanismy na základě sítě (*network-based*) jsou v případě SDN nejpočetněji obsažená skupina. Obecně se mohou nacházet na směrovačích pro daný autonomní systém. Dále se mohou dělit na *attack specific* a *anomaly based*. První typ se zaměřuje na určitý útok, například na konkrétní službu a druhý typ pracuje na principu porovnávání stavu sítě s prahovými, či obvyklými hodnotami.

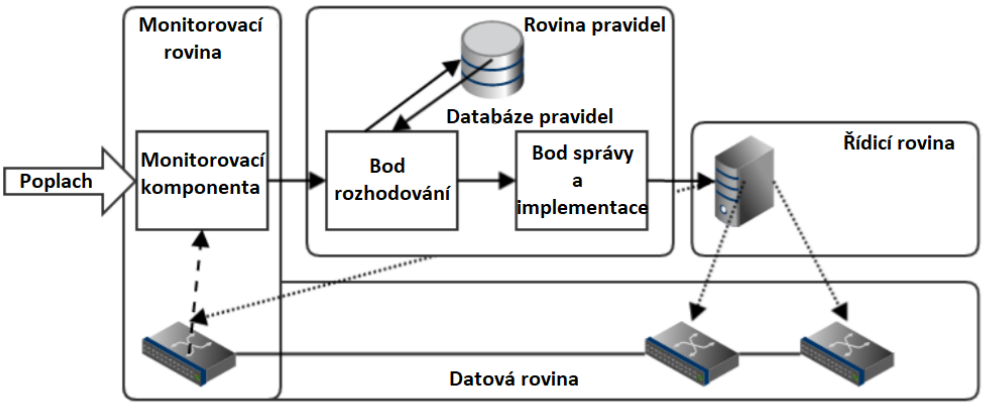
3. Mechanismy na základě cíle (*destination-based*) spoléhají na to, že odhalení a řešení útoku proběhne na straně cíle, a to často pomocí rekonstrukce či sledování cesty, odkud pakety přicházejí. Mezi časté způsoby implementace patří pravděpodobnostní označování paketů, *input debugging*, či *IP traceback*.

4. *Hybridní* mechanismy či jejich komponenty jsou rozloženy ve všech třech předchozích místech, přičemž spolu úzce spolupracují.

Řešení DDoS na straně poskytovatele konektivity

Práce [38] popisuje rámec, který umožňuje dynamickou správu pravidel, pomocí nichž můžou ISP (*Internet Service Provider*) automaticky zmírnit dopad útoků na síť. Dokáže tak ve velmi krátkém čase omezit další vzniklé škody a přetížení v sítích zákazníků. Navrhovaný rámec pro správu a uplatnění pravidel odlišuje požadavky konkrétních zákazníků a neovlivňuje tak provoz dalších.

Autoři zdůrazňují potřebu provádět *traffic engineering* nejen na straně ISP. V případě obrany proti „*DDoS*“, který má za cíl vyčerpat zdroje ISP, nestačí pouze zvýšit prioritu legitimního provozu, či přesměrovat podezřelý provoz daného zákazníka. Za předpokladu sdílení kapacit mezi více zákazníky by tak docházelo k současnému ovlivnění ostatních. Dále předkládají narůstající potřebu spolupráce mezi ISP a zákazníkem, který jinak nemá mimo blokování útoku na svém perimetru moc možností. Proto je v zájmu obou spolupracovat na řízení provozu společně, díky čemuž také může zákazník lépe definovat svoje požadavky a ISP je tak snadněji realizovat. Dosud navrhovaná řešení bez úzké spolupráce obou stran tak přináší zbytečně dlouhé prodlevy ve zjištění a samotné reakci na útok.



Obrázek 2.8 - Architektura na straně ISP, upraveno z [38]

Obrázek 2.8 popisuje ideové schéma komponent. *Monitorování* spravuje možné poplachy od zákazníků a sleduje stav zařízení a cest v síti ISP. Získané informace zpracovává pro *bod rozhodování.* Na základě získaných informací od *Monitorování* a *databáze pravidel* rozhoduje jakou akci podniknout. Vybrané akce a informace o konkrétních „flow“ jsou předány na *bod správy a implementace*, který vybírá cesty a dále za pomocí SDN distribuuje pravidla na konkrétní zařízení v podobě záznamů „flow“ tabulek. *Databáze pravidel* obsahuje definice bezpečnostních pravidel, které jsou specifikovány správci sítě. V případě zjištění podezřelého toku tak může dojít v závislosti na zadaných pravidlech například ke směrování dat na „*firewall*“.

Obrana pohybujícími se cíli

Obrana pohybujícími se cíli *(moving target defense)* je způsob jakým zmenšit povrch útoku a zmást útočníky. Nejčastěji se jedná o časté změny jinak statických atributů, jako jsou přidělené IP adresy, porty aj. [39] Díky tomu se daří útočníky zdržet, popřípadě jim kompletně zmařit pokus o útok.

Mapování sítě je často označován jako první krok při přípravách samotného útoku, kdy se útočník snaží získat informace o síti, jako třeba verze operačních systémů a běžících služeb, možná úzká hrdla či další adresy v síti pro šíření červů aj.

Skenování jsou prováděna například pomocí ICMP zpráv, kterými se ověřuje dostupnost a konektivita. TCP a UDP zprávami pro sledování otevřených portů a spuštěných služeb nebo třeba polem TTL, ze kterého útočníci určují počet přeskoků k cíli.

Autoři [39] popisují snadno implementovatelný způsob, kterým výrazně snižují efektivitu mapování sítě a služeb. Díky SDN a navrhovanému algoritmu tak může být provoz na základě definovaných pravidel tajně zahozen a místo něj vygenerovány měnící se odpovědi, které mohou útočníka zmást. Provedená akce nad pakety je dočasně udržována ve vyrovnávací paměti pro konzistentní odpovědi. Algoritmus odhaluje další zavřené i otevřené porty a posílá zprávy *ACK, PUSH-ACK* pro zmatení pokročilejších skenovacích nástrojů. V případě ICMP se jedná o informování o náhodně spouštěných zařízeních a mezilehlých přeskocích, které ve skutečnosti neexistují. Ve výsledku tak hlubší analýza stojí útočníka další čas a výkon.

Obrana pohybujícími se cíli a *DDoS*

Autoři [40] navrhují řešení s využitím SDN ve vysokorychlostních sítích používaných mezi ISP. Poukazují na důležitý aspekt, kterým je potřeba spolupráce daných správců autonomních systémů a sdílení informací o bezpečnostních incidentech. Mezi hlavní komponenty testovaného řešení patří monitorování „*flow*“, výměna informací, spolupráce s ostatními ISP a síťový operační systém ONOS. Práce bere v úvahu obranu pohybujícími se cíli na úrovni sítě a hosta. Ta na úrovni sítě je založena na různých BGP cestách, které jsou předpřipraveny na hraničních routerech a jsou nastaveny tak, aby měnily tvar sítě. Na úrovni hosta se jedná o „*IP-hopping“*, přičemž dochází k využívání tzv. „*honeypotu“*.

V práci autoři popisují kritéria pro hodnocení efektivity *DDoS* obrany a na jejich základě dále svoje řešení hodnotí. *Diverzifikace* popisuje možnost výběru jedné z mnoha dostupných konfigurací sítě. Ta přirozeně roste s velikostí sítě. Schopnost *adaptace* popisuje možnost změny tvaru grafu sítě při zachování všech funkcionalit. Nepředvídatelnost pohybu a transformací popisuje *náhodnost*. *Entropie* popisuje efektivitu dané strategie. *Komplexnost nasazení* popisuje, jak náročná je implementace daných řešení a technologií v síti. *Včasnost* popisuje náročný problém provedení adaptace v přijatelném čase. Dalším kritériem je *schopnost vyvažování zátěže* a odvedení provozu například do „*honeypotu“*, či k další analýze. V neposlední řadě sem patří *škálovatelnost a cena*.

2.9 SDN a mobilní operátoři

Jednou z hlavních výhod SDN pro operátory je větší přenositelnost a škálovatelnost. Do nedávna byli nuceni neustále rozmisťovat dedikované boxy do různých částí sítě. To přináší problémy s přidáváním nových zařízení, plánováním, rozšiřováním kapacit či správou.

Řízení virtualizovaného jádra sítě

Článek [41] popisuje práci skupiny ONF na implementaci SDN do MPC (*Mobile Packet Core*) a na jejím základě navrhuje architekturu MPC s dynamickým zajišťováním služeb a s využitím SDN a NFV. Jedná se tak o přirozenou evoluci, kde jsou jednotlivé funkce jádra virtualizovány. Pomocí SDN dochází k oddělení kontrolní a uživatelské roviny a také oddělení řízení síťových služeb a řízení provozu.

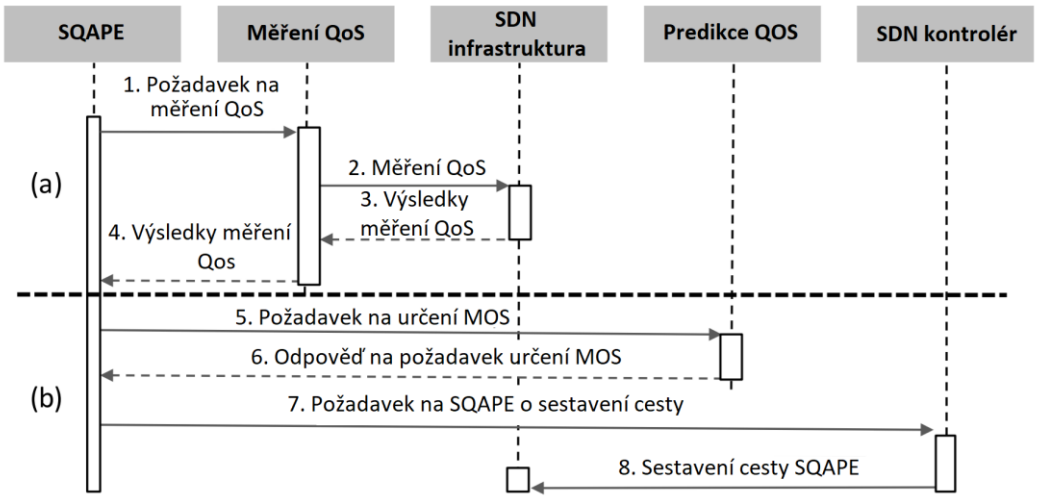
Práce řeší řídicí rovinu logicky centralizovaného EPC (*Evolved Packet Core*) jako skupiny virtualizovaných služeb provozovaných v datových centrech operátora. *SDN kontrolér pro NFV doménu* se stará o dostupnost výpočetního výkonu a uložiště konkrétní části infrastruktury v datovém centru. *Kontrolér end-to-end konektivity* je hlavním bodem, který se stará o směrovací rovinu a její funkce, jako jsou například vyvažování zátěže, správa topologie nebo správa výpočetního výkonu. Stará se o pravidla v jemu podléhající síti. Ty se neustále mění na základě vytížení, pozice UE (*User Equipment*), QoS aj.

Správa NFV má na starost virtuální sítě v datovém centru, konfigurace VNF a jejich rozhraní. Například při připojení nového MME (*Mobility Management Entity*) jej nakonfiguruje a asociuje s dalšími funkcionalitami jako jsou PGW (*Packet Data Network Gateway*), HSS (*Home Subscriber Server*) či další MME. Dále komunikuje s SDN kontroléry pro zajištění konektivity nové funkcionality.

Centralizované řídící funkce musí současně pracovat s různými zdroji. Práce řeší možnost oddělení řízení síťových služeb a řízení provozu díky rozdělení na dvě spolupracující SDN vrstvy. Například směrování mezi PGW a SGW zajišťující „*end-to-end*“ konektivitu uživatele k Internetu patří mezi řízení provozu. Mezi zdroje potřebné pro samotný chod a služby sítě patří přístup do databází, ukládání *logů*, zápisy do registrů, bezpečnost, alokace výpočetního výkonu, NAC aj.

Škálovatelné řešení výběru cesty na základě parametru QoE

Neustále narůstající požadavky uživatelů na přenosové parametry sítě vedou operátory k rozkládání zátěže pomocí technologií, jako jsou CDN (*Content Delivery Networks*), „*edge computing*“, či D2D komunikace. To zvyšuje efektivitu v podobě zkracování vzdáleností k účastníkovi, ale zároveň přináší řadu alternativních cest, mezi kterými je nutné neustále vybírat. Autoři [42] popisují nasazení škálovatelné technologie SQAPE pro efektivní směrování video provozu v rozsáhlých sítích operátorů, při udržování vysokého QoE (*Quality of Experience*).



Obrázek 2.9 - sekvenční diagram algoritmu SQAPE, upraven z [42]

Architektura SQAPE se skládá ze tří hlavních komponent: měření QoS, predikce QoE a rozhodovací blok. Popsaný algoritmus bere v potaz dva důležité parametry. Při hledání cest se snaží o rovnováhu mezi maximálním MOS (*Mean opinion score*) a minimálními nároky na prostředky. Hledání maxima MOS je popsané jako *widest path problem* a minimalizace využitých prostředků jako *minimum path problem.*

Obrázek 2.9 popisuje start procesu, kdy události 1-4 jsou periodicky spouštěny na základě četnosti intervalů monitorování QoS. Na základě aktivního měření šířky pásma, zpoždění a ztrátovosti předávají výsledky naměřeného QoS. Aktivní měření je využíváno z důvodu častého výskytu této měřící služby v síti operátorů.

Události 5-8 jsou spouštěny na základě požadavků uživatele o dané video toky. Dochází ke skládání QoS metrik a spuštění mapovací funkce *QoS-to-QoE* pro spočítání odhadu MOS pro každý spoj. Spolu s aktuálním vytížením spoje je tento odhad využit k výpočtu metriky, na základě které je rozhodováno o směrování. Díky měření jednotlivých spojů na místo všech možných kombinací cest je tento postup méně náročný na režii a výpočetní výkon, čímž je lépe škálovatelný.

3 Současné možnosti implementace SDN a NFV

3.1 Hardware

Ačkoliv jsou zařízení postavená dle filozofie otevřenosti SDN často souhrnně označována jako přepínače, jedná se o chytrá zařízení, která jsou nejen díky kontroléru schopna obstarávat služby vyšších vrstev, jako je například IP či BGP směrování nebo funkcionality *„firewallu“*.

Mezi největší výrobce čipových sad, které umožnily nástup zařízení podporujících SDN, patří například firma *Broadcom*. Ta během několika let představila sérii *Strata XGS Tomahawk*, která je v současné době již v třetí generaci. Další úspěšná řada je nazvaná *Trident* a její současná verze *Trident II*. Dalšími výrobci jsou *Juniper* s čipovou sadou *Penta* nebo významný čínský výrobce *Centec*. [43]

Následující Tabulka 3.1 uvádí souhrn zástupců HW řešení pro SDN ze všech segmentů a typů sítí. Pro přehlednost nejsou uvedeny všechny možné konfigurace či výrobci, stejně tak byly vynechány starší generace zařízení a výrobci, kteří již ukončili svůj prodej či vývoj. Většina přepínačů podporuje různé, ať už otevřené či čistě proprietární kontroléry, přičemž ale všechny podporují alespoň OF ve verzi 1.3 a vyšší. Tabulka byla zpracována na základě katalogových listů a webových stránek výrobců.

Pro často využívanou dvou úrovňovou architekturu sítě v datových centrech se vžilo označení list-páteř *(„leaf-spine“)*. Listové přepínače agregují provoz od jednotlivých serverů směrem do jádra, které předchází druhá vrstva páteřních přepínačů. Ty aktivně komunikují s kontrolérem a jsou dále připojeny na samotné jádro sítě. Tento návrh přináší snadnou škálovatelnost a robustnost v podobě redundantních spojů listových přepínačů k většímu počtu páteřních přepínačů. Díky využití vysokokapacitních optických kabelů je značně zredukováno množství kabeláže. ToR (*Top of Rack*) je označení pro přepínač, který slouží k agregaci provozu ze serverů v konkrétním racku a zastává tak totožnou funkci jako list.

Tabulka 3.1 - Dostupné přepínače

Výrobce	Modelová řada	Možné Konfigurace	Použití
Accton	VED2181F	1x RJ45 (1000BASE-T) uplink 4x RJ45 (1000BASE-T) LAN	„*virtual edge*“ zařízení
EdgeCore	AS6712-32X	32x QSFP+ (40GbE) (96 x 10 GbE + 8 x 40GbE)	ToR / páteřní přepínač
EdgeCore	AS7700-32X	32x QSFP28 (40/100 GbE) (64x 50 GbE) (128x 10/25 GbE)	ToR / páteřní přepínač

Arctica	3200cs	32x QSFP28 (32x 100 GbE) (64x 50 GbE)	ToR
Arctica	4806xp	48x SFP+ host portů (10 GbE) 6x QSFP uplink portů (40 GbE)	ToR přepínač
Celestica	D4040	32x QSFP (40GbE)	ToR přepínač
Quanta Computer	QuantaMesh T5032-LY6	32x QSFP (40GbE)	ToR / páteřní přepínač
Quanta Computer	QuantaMesh BMS T7032-IX1	32x QSFP (10/25/40/50/100GbE)	ToR / páteřní přepínač
Agema / Delta Networks	AG9032 v2	32x QSFP28 (100GbE) 2x SFP+ (10GbE)	Páteřní přepínač
Agema / Delta Networks	AG 5648	48x SFP28 (25GbE downlink) 6x QSFP28 (100GbE uplink)	Listový přepínač
Juniper	PTX10008 a PTX10016	QSFP28, QSFP+, QSFP56DD či DWDM (Dle konfigurace až 288x, resp. 576x 400 GbE)	Jádro / víceúčelový
Juniper	MX2020	QSFP28, QSFP+, či QSFP56DD (Dle konfigurace až 800x 100GbE, 320x 200GbE nebo 160x 400GbE)	List / hranice sítí WAN / jádro sítě
Noviflow	NoviSwitch 2128	4x QSFP+ (40GbE uplink) 24x SFP+ (10GbE downlink)	Listový přepínač
Noviflow	WB-5164	64x QSFP28 (100GbE)	Hranice sítí WAN / jádro sítě
Ciena	5162	2x QSFP28 (40/100GbE) 40x SFP+ (10 GbE)	Hraniční směrovač
Ciena	Z-series (Z77)	SFP+, XFP, CFP, technologie SONET/SDH, OTN či DWDM (až 200Gbps / slot)	Optická platforma pro jádro regionálních sítí
Nuage Networks	210 WBX 32QSFP28	32x QSFP28 / QSFP+ (100GbE)	Listový přepínač

Nuage Networks (Nokia)	7850	6x RJ45 (1000BASE-T)	Propojení datacenter s pobočkami
Lenovo	ThinkSystem NE10032O RackSwitch	32x QSFP+/QSFP28 (32x 100 GbE)	ToR přepínač
Lenovo	RackSwitch G8296	86x SFP+ (10 GbE) 6x QSFP+ (40 GbE)	Listový / páteřní přepínač
Arista (HP)	7060(D)X4-32	32x OSFP (QSFP-DD) (100 GbE / 400 GbE)	Listový / páteřní přepínač
Cisco	3432D 3408	32x QSFP-DD (400 GbE) 128x QSFP28 (100 GbE) nebo 32x QSFP-DD (400 GbE)	ToR / páteřní přepínač
Cisco	Nexus 9316D 9360CD	16x QSFP-DD (400 GbE)	Listový přepínač
Cisco	Nexus 9360CD	28 x QSFP28 (100 GbE) 8x QSFP-DD (400 GbE)	List / hranice sítí WAN / jádro sítě
DellEMC	S4248FBL-ON	40x SFP+ (10 GbE) 2x QSFP+ (40 GbE) 6x QSFP28 (100 GbE)	ToR
Dell EMC	Connectrix B-Series, Connectrix MDS Series, Connectrix D-Series	8- 96x SFP+ (10 GbE) 6x QSFP (40 GbE)	ToR přepínač
Pica8	P-5401	32x QSFP+ (40GbE)	Listový / páteřní přepínač
Pica8	P-3922	48x SFP+ (10GbE) 4x QSFP (40GbE)	Listový / páteřní přepínač
Extreme networks	Black Diamond X8	Podle konfigurace: 1 152x RJ45 (1000BASE-T) / 576x QSFP+ (40 GbE) / 96x CFP2 (100 GbE)	Listový / páteřní přepínač
Plexxi	Plexxi Switch 2e ENTRY SERIES	6x QSFP (40 GbE) 48x SFP+ (10 GbE)	Listový přepínač
Centec	V580-48X2Q4Z	48x SFP+ (10 GbE) 2x QSFP+ (40 GbE) 4x QSFP28 (100GbE)	Listový přepínač

NEC	PF5240	4x SFP+ (10 GbE) 48x RJ45 (1000BASE-T)	Listový přepínač
NEC	PF5340-32	32x QSFP+ (40GbE)	Univerzální použití
Brocade	MLX- 4–32 4-32 chassis system	Podle konfigurace: 64x 100 GbE, 128x 40 GbE, 768x 10 GbE na jeden slot	ToR / páteřní přepínač
NetGear	M4300 series	Podle konfigurace: od několika 10GBASE-T po stovky SFP+ (40GbE)	Listový přepínač
Huawei	AR3600 series	Podle konfigurace: VDSL2, GPON, EPON, RJ45 1000BASE-T	Přístupové sítě PON
Facebook	Wedge-100	32x QSFP28 (100GbE)	TOR přepínač
Facebook	6-pack	128x 40GbE	Otevřená modulární přepínací platforma

3.2 Operační systémy síťových prvků

SwitchLight OS vznikl jako komerční odnož *Open Network Linux*, který provedla firma *BigSwitch Networks*. Dále nabízí verzi VX, jakožto agenta pro kernel virtualizovaných přepínačů v *Open vSwitch*. [44]

OpenSwitch OPX Base je zástupcem OS s otevřeným zdrojovým kódem pro síťové prvky od *Linux Foundation.* Mezi hlavní podporovatele patří firma *Dell*. Je postavený na neupraveném jádru *Debian Jessie*. Předpřipravená API umožňují snadný vývoj dalších aplikací ať už v *Pythonu*, C, či C++. [45]

Mezi další operační systémy patří **Snabb**, **FBOSS**, **Open Network Linux**, které jsou dále popsány v kapitole 3.7.

3.3 Kontroléry

NOX je prvním vzniklým SDN kontrolérem s otevřeným zdrojovým kódem. Pochází z *Nicira Networks* (dnes patřící pod *VMWare*), kde byl vytvořen pro práci s *Openflow.* Obsahuje základní komponenty pro správu topologie či nalezení nových přepínačů. Byl napsán v C++ a sloužil jako významný vývojový prvek v počátcích SDN. Umožňuje psát uživatelské aplikace v C++ či *Pythonu*. Na jeho základě byly dále odvozeny další známé kontroléry. V současné době není dále vyvíjen a jeho poslední aktualizace byla v roce 2014. **NOX-MT** je mírně upravený NOX s podporou více vláken, který díky jednoduchým úpravám jako jsou například „*I/O batching*“ dokáže výrazně zlepšit odezvu a propustnost nabízenou předchozím NOX. [46]

POX je dalším následovníkem NOX, přičemž může fungovat i jako samotný OF přepínač. Byl napsán v jazyce *Python* a nabízí více předpřipravených komponent pro výběr cest, správu topologie či rozšíření pro podporu *Open vSwitch* aj. Mimo OF podporuje i OVSDB protokol. Stal se populárním díky svým uživatelsky přívětivým rozhraním. [47]

Beacon je kontrolér napsaný v *Javě*, který za pomoci knihovny *Spring* nativně podporuje více vláknové zpracovávání. Oproti předchozím nabízí webové rozhraní a možnost dynamického spouštění, zastavování nebo instalování nových aplikací za běhu hlavního procesu kontroléru. [47]

Floodlight vznikl jako odnož *Beaconu* a později se stal základem pro jedny z prvních komerčních kontrolérů od *BigSwich Networks*. Představuje zástupce kontrolérů umožňujících fungovat i ve smíšených sítích obsahujících jak OF zařízení, tak zařízení klasické. Podporuje severní rozhraní REST, což usnadňuje další vývoj aplikací.

SE-Floodlight je rozšíření snažící se o zvýšení celkové bezpečnosti kontroléru implementací autorizace na základě rolí, řešení konfliktů pravidel, či službu bezpečnostní audit. Přesto i se všemi ostatními běžícími na jedné instanci zůstává určitým nebezpečím jakožto SPOF (*single-point of failure*) a možnost úzkého hrdla. Díky svojí funkcionalitě a výkonu se přesto *SE-Floodlight* stal základem pro distribuované architektury SDN jako je například ONOS. [48]

RYU vydaný pod licencí *Apache* 2.0 je kontrolér napsaný v *Pythonu* a velkou výhodou je plná podpora OF ve verzích 1.0 až 1.5. Jeho základní filozofií je schopnost fungovat jako modulární rámec obsahující různé stavební kameny, namísto rozsáhlého kontroléru. [49]

OpenDaylight Platform je výsledek práce *OpenDaylight foundation*, která je součástí *Linux Foundation*. Kontrolér je napsaný v *Javě*, přičemž svoje kořeny má v kontroléru *Beacon*. Díky otevřenosti, spolupráci významných subjektů a zaměření na komerční využití je tento kontrolér velmi propracovaný a pokročilý. Podporuje řadu rozšíření, protokolů a rozhraní, mezi kterými nechybí například integrace s *OpenStack*, jazyk P4 (*Programming Protocol-independent Packet Processors*), podpora virtualizace v různých prostředích za pomocí kontejnerů a mnoho dalších. Jeho současnou verzí, které jsou pojmenovávány vždy podle chemických prvků, je verze Fluor. [50]

Cisco ACI (*Application Centric Infrastructure*) je i přes některé odlišnosti považován za SDN řešení, určené primárně pro datová centra a prostředí „*cloudu*“. HW řešení je postaveno na proprietárních přepínačích série *Nexus* 9000, přičemž podporuje i virtuální přepínače *Cisco* AVS (*Application Virtual Switch*). *Cisco* APIC (*Application Policy Infrastructure Controller*) slouží jako beze stavová autorita pro správu a konfiguraci pravidel. Dochází tak k rozvázání konfigurací s jednotlivými zařízeními. APIC zastává centrální úložiště všech pravidel. Jako protokol jižního rozhraní využívá *Cisco OpFlex*. I přes svoje proprietární řešení je ACI otevřené službám třetích stran jako jsou *SourceFire*, *Embrane* či *Citrix*.

3.4 Síťové operační systémy

ONOS je významný, otevřený, komplexní síťový operační systém, který vznikl na základě práce inicativy *OpenDaylight*. Původně byl cílený na provozovatele transportních sítí a obsahoval podporu pro CORD (*Central Office Re-architected as a Datacenter*) od původních vývojářů *ON.lab*. Ten se snaží o přesun klasických *„central offices“* na straně provozovatelů v model bližší těm nasazených v datových centrech. Dnes v sobě navíc obsahuje aplikace pro další široká využití jako například *„edge computing“* nebo PON.

ONOS je založený na modulární architektuře, podporuje široké možnosti protokolů pro jižní i severní rozhraní a díky distribuovanému přístupu tak zaručuje vysokou dostupnost služeb. [51]

HP VAN je další komplexní NOS napsaný v *Javě*, který nabízí řadu aplikací ať už pro správu datových center, veřejných i privátních *„cloudů“*, ale i například přístupových sítí v kampusu.

Jedná se o uzavřené řešení firmy HP, na které se vztahují licenční poplatky. Díky využití kontejnerů je tato modulární platforma snadno přenositelná a flexibilní. Mimo jiné podporuje REST API, přes nějž nabízí přehledné webové rozhraní. Jakožto komerční řešení nativně podporuje zabezpečené jižní rozhraní pomocí TLS. HP dále nabízí svoji produktovou řadu SDN zařízení, které ale díky podpoře OF nejsou podmínkou pro využívání tohoto kontroléru. [52]

Další software, často ne velmi přesně zařazovaný mezi NOS, je spíše sérií návrhů a výzkumů, které se zaměřují na konkrétní problémy, nevedou k přímo využitelnému produktu, byl ukončen jejich vývoj, či nejde o plně distribuované NOS. Mezi tento SW patří například:

Hyperflow je první projekt snažící se rozšířit OF o fyzicky distribuovanou řídící rovinu. Umožňuje spustit vícero instancí NOX v síti a zaručit jejich synchronizaci. Byl implementován jako aplikace napsaná v C++ pro kontrolér NOX. Výhodou jsou minimální změny v kódu NOX a možnosti použití jeho aplikací. [53]

Projekt **Rosemary** předkládá časté chyby ve vývoji aplikací a implementaci SDN, přinášející možnou nestabilitu, pády, či zranitelnosti. Dále navrhuje vlastní řešení kontroléru s vyšší odolností na základě omezení jednotlivých aplikací. Každá aplikace je spuštěna v rámci své *Rosemary* instance, které jsou monitorovány a jsou jim přiřazovány výpočetní zdroje na základě oprávnění, či priority. [54]

Fleet navrhuje architekturu pro potlačení zlomyslných praktik a distribuci špatných konfigurací ze strany administrátorů a případné zotavení sítě. [55]

DISCO je kontrolér postavený na *Floodlightu*, který se zaměřuje na překryvné a WAN sítě. Návrh popisuje kanál pro komunikaci ve východním a západním směru mezi DISCO kontroléry v jednotlivých sítích provozovatelů. [56]

3.5 Programovací jazyky

Práce [57] přináší komplexní třídění programovacích jazyků pro SDN. Jako hlavní dělení uvádí 3 hlavní skupiny:

V případě *nízko úrovňového programování*, jsou síťové prvky programovány přímo za pomoci *control to data plane* rozhraní. Tento přístup nabízí omezenou nabídku konstruktů a komplexnější aplikace obsahují velké množství řádků kódu, přičemž chyby či nekonzistentní stavy musí být ošetřeny samotným programátorem. Nejčastěji sem patří GPL (*General-purpose Languages*) jako jsou *Java*, C či *Python*. Program tak definuje samotnou posloupnost *Openflow* zpráv, včetně potřebných parametrů a zpráv.

U *programování na základě API* jsou aplikace psány s využitím API, které jsou jim poskytovány kontrolérem. Ten je dále překládá na zprávy. Dochází však ke stejným omezením jako v případě nízko úrovňového programování. Opět zde vzniká monolitický kód, který není snadné „*debutovat*“ či znovu použít.

Kontroléry zpravidla nabízí 2 druhy API. Program využívající *lokální API* musí běžet na stejném hostu jako kontrolér, využívat pouze jeho třídy a být napsán ve stejném jazyce. *Vzdálené API* vytváří určitou úroveň abstrakce mezi rozdílnými prostředími či programovacími jazyky, přičemž nemusí běžet na stejné instanci jako samotný kontrolér. Nejčastěji využívají pro přenos informací XML či JSON.

Třetí skupinou, která nabízí nejvyšší úroveň abstrakce, jsou programy v doménově specifickém (*domain-specific)* jazyce, které jsou zaměřené na určitou oblast. Přináší modulární kompozici, abstrakci topologie, možnost virtualizace, či monitorování pro automatizované programování přepínačů. Tyto jazyky obsahují vlastní překladač, který překládá vysokoúrovňové konstrukty do formy, která je srozumitelná API kontrolérů. Mezi tyto jazyky patří například:

Frenetic je jedním z prvních doménově specifických jazyků, který vznikl. Později se stal základem mnoha dalších jako jsou například *Merlin, FatTire, FlowLog, Pyretic, Kinetic* aj. Jedná se o deklarativní jazyk podobný SQL dotazům, pro klasifikaci, sledování či agregaci provozu. Obsahuje knihovny pro definici směrovacích pravidel. Mezi dostupné konstrukty patří možnost seskupování informací podle času, na základě plovoucího okna, či podle počtu paketů nebo jejich velikosti. [57]

Pyretic je imperativní jazyk napsaný v Pythonu vycházející z projektu *Frenetic*. Poskytuje možnost současného sekvenčního i paralelního zpracování více pravidel najednou. Přináší možnost abstrakce paketů, která umožňuje přidání virtuálních polí v hlavičkách paketů. Dále v sobě obsahuje knihovnu pro abstrakci topologie, díky které je možné například sjednotit několik přepínačů pod jedním virtuálním. Překladač napsaný program překládá na instrukce pro vzdálené API projektů, které podporují *Python* jako například NOX či *Ryu*. [58]

Merlin je rámec pro programování síťových pravidel v SDN. Stará se o distribuci a koordinované prosazování pravidel. Obsahuje konstrukty pro určení maximální či garantované šířky pásma. Dále přináší monitorování, které je schopné dohlížet na provoz koncových uživatelů. Jako jeden z prvních poskytl nájemcům možnost za pomoci „sub-pravidel“ zasahovat do směrovacího rozhodování, přičemž hlídá, aby nebyly porušeny globální pravidla nastavená správcem. [59]

Procera také udává možnost zahrnout další externí události získané mimo OF do tvorby pravidel. Umožňuje tak schopnost reagovat například na vytížení serveru, denní dobu, či autentifikaci uživatele v systému. Stejně jako například *Frenetic* dokáže využívat časové okno, avšak nejen na samotná „*flow*“, ale i na tyto externí události. [57]

3.6 Jižní rozhraní

OVSDB je dalším významným jižním API, které nabízí pokročilé možnosti správy. Vznikl jako součást projektu *Open vSwitch*. Oproti *Openflow* poskytuje navíc možnost řídit QoS na jednotlivých rozhraních, přidávat rozhraní, síťové mosty, konfigurovat tunelování na OF cesty, vytvářet více instancí virtuálních přepínačů aj. Zaměřuje se spíš na samotnou konfiguraci přepínače a tyto dvě technologie jsou tak často využívány společně.

Cisco OpFlex představuje zástupce protokolu využívaného v rodině produktů *Cisco* ACI. *OpFlex* se drží konzervativnější cesty a oproti OF nevolí kompletní centralizaci všech funkcí na kontroléru, ale ponechává určitou formu inteligence na samotných síťových zařízeních. Jako hlavní důvod je označována snazší škálovatelnost a větší dostupnost. Jedná se o protokol umožňující obousměrnou komunikaci, přičemž definice pravidel je možná ve formátech JSON či XML. Díky otevřenosti a vývoji *Opflex* agenta je možné tento protokol využívat i na jiných platformách jako například *OpenDaylight*. [60]

3.7 Současné trendy

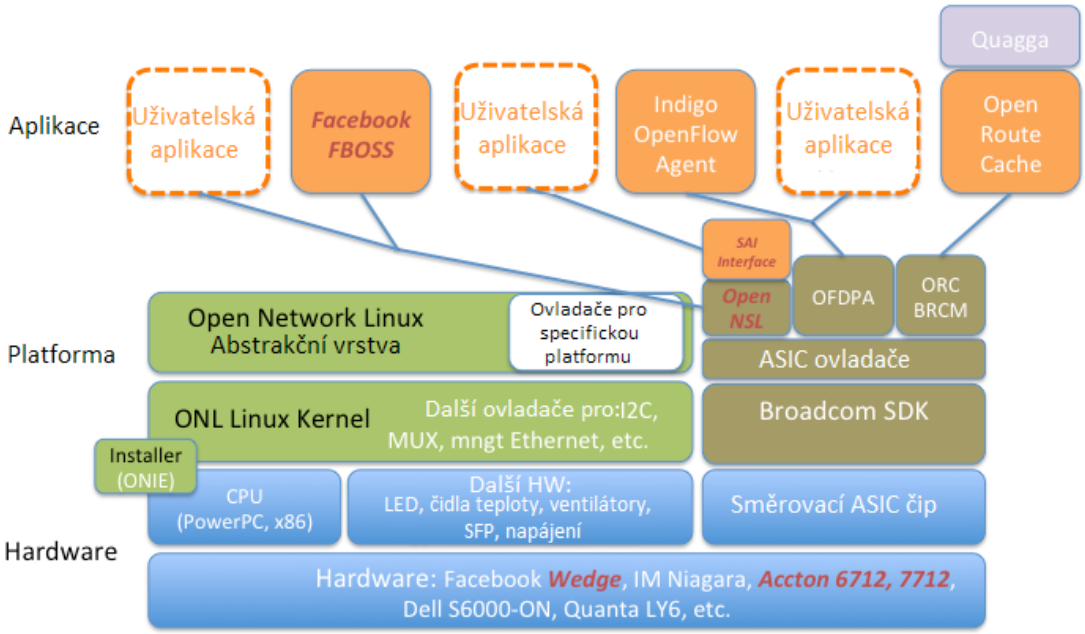
P4 (*Programming Protocol-independent Packet Processors*) je slibným projektem programovacího jazyka, který se později přidal pod ONF a je tak podporován ve všech jejich dalších produktech. Jeho syntax je velmi podobný jazyku C. Hlavním důvodem pro vznik byla potřeba dosáhnout nezávislosti jak protokolů, tak platformy. Toho je dosaženo pomocí rozlišení dvou režimů práce. Během konfigurace zařízení se do něj nahrává program a interně se překládá. To znamená, že program je přeložený pro dané zařízení s ohledem na jeho vnitřní architekturu. Po naprogramování přechází do normálního provozu, kdy dochází k mapování akcí na toky. [61]

NoF *(network overlay framework)* je dalším zástupcem rámce pro programování SDN. Přináší univerzální modul, který je schopný překladu programů z NoF do API většiny kontrolérů. Dále dokáže číst stavy a informace z koncových zařízení, jako například vytížení systému, název systému, jméno procesu aj. [57]

POF *(protocol oblivious forwarding)* vznikl jako konkurence k *Openflow*. Mezi hlavní důvody jeho vzniku patří problematika protokolové uzavřenosti OF. Ten pro svoji funkci vyžaduje jasně stanovené pole hlaviček a jejich hodnoty, což je při implementaci nových, nestandartních protokolů velmi omezujícím faktorem. Totéž platí pro akce, které jsou v OF pro dané protokoly definovány. Dalším problémem je zpětná kompatibilita jednotlivých verzí OF, kde v každé nové verzi přibývají další podporované protokoly a akce nad nimi. POF přináší nezávislou sadu instrukcí pro definice protokolů a zpracování packetů. Vyhledávání v polích probíhá na základě obecného „*tuple*“ <odsazení, délka>. Návrh v sobě obsahuje i možnost komunikace ve směru východním a západním a využívá k tomu zprávy ve formátu JSON, na základě kterých si pak kontroléry vytváří globální virtuální topologii podobně jako například protokol IS-IS. Toto API je v praxi podporované firmou *Huawei*. [62]

Open Network Linux se stal největším otevřeným projektem organizace *Open Compute Project*. Jedná se o Linuxovou distribuci založenou na *Debianu*, která je určená pro SDN přepínače. Poskytuje potřebnou abstrakci a umožňuje tak využívat tento OS na více jak 68 různých HW platformách.

Podporuje softwarové balíčky, jako například *Quagga routing suite, Facebook Open Switching System* či *Openflow Data plane Abstraction*. Tyto balíčky přináší implementované směrovací protokoly, jako jsou OSPF, BGP, IS-IS nebo třeba podporu pro *Openflow*. [63]



Obrázek 3.1 - Architektura Open Network Linuxu, upraveno z [64]

Snabb je otevřená sada nástrojů, která virtualizuje *ethernetovou* sadu. Umožňuje tak manipulovat s rámci ve virtualizovaném přepínači. Díky využití skriptovacího jazyka *Lua* a předpřipraveným aplikacím, jako například paket filtr, IPv4 přes IPv6, generátor provozu a jiné, je snadné doplnit síťová zařízení o nové funkcionality. [65]

FBOSS je podobně jako *Snabb* sada aplikací, které je možné spustit na standartních distribucích *Linux*. Jedná se o SW využívaný v datových centrech *Facebooku*, který byl později uvolněn jako otevřený zdrojový kód pro další vývoj. Mezi uváděné aplikace patří *FBOSS agent*, což je aplikace pro komunikaci se síťovým zařízením a pomocí přidané abstrakce usnadňuje vývoj síťových aplikací. Dalšími aplikacemi jsou konfigurační nástroje, monitorování a řídící aplikace implementující směrovací protokoly. [66]

Stratum je novým projektem ONF v čele se společností *Google*, jehož koncept byl představen v roce 2018. Jejich cílem je uvedení nových referenčních řešení a to „*whitebox*“ síťových prvků spolu s novým operačním systémem pro přepínače. Hlavní motivací je urychlení vývoje a nasazení v sítích. První verze, založené na kódu od Google, by měly vyjít pod licencí *Apache* 2.0. Jednou ze součástí projektů je potenciální nástupce *Openflow*. Nejen díky využití jazyku P4 či gNMI (*Google Network Management Interface*) by měl *Stratum* umožnit zasahovat do směrování jako OF, ale i do definicí „*pipeline*“, jejich řízení a konfigurací, což přináší kompletní kontrolu daného zařízení. [67]

4 Doporučená řešení za pomoci SDN

Následující tabulka 4.1 vznikla během přípravy předcházejících dvou kapitol jako souhrn současných výzev a problémů v oblasti sítí. Dále navrhuje jejich možné řešení v prostředí softwarově definovaných sítí. Uvažovaný HW je závislý na požadovaných počtech rozhraní, rychlostech či třeba preferenci dodavatele.

Tabulka 4.1 - Souhrn síťových problémů a jejich řešení za pomoci SDN

Možná síť (typ/topologie)	**Požadované parametry a možné problémy**	**Výhody nasazení SDN**	**Příklady použité technologie**
CDN (*Content delivery Network*)	Vyvažování zátěže a redistribuce toků Správa velkého množství zařízení Škálovatelnost Komplexní QOS Šířka pásma a řízení přetížení	Přímá ovlivnitelnost řídící roviny Centrální a jednotné řízení prvků a jejich konfigurace	Juniper MX2020 Cisco 3432D DellEMC S4248FBL-ON
Cloud/datové centrum	Vyvažování zátěže a redistribuce toků Správa velkého množství zařízení Škálovatelnost Řízení TOR přepínačů Komplexní QoS Šířka pásma a řízení přetížení	Abstrakce a globální přehled o sítí Neutralita výrobců a dodavatelů Rychlé prototypování Virtualizace ASIC zařízení	Arctica 4806xp Delta Networks AG5648 Extreme Networks Black Diamond X8
„X“ jako služba	Virtualizace sítí a řetězení NFV *„Multi-tenant“* architektury Bezpečnost jako služba	Spojení více zařízení do jednoho celku (IDS, *„firewall“,* vyvažování zátěže…) Automatizace nasazení služeb a rekonfigurace	Noviflow WB-5168 Ciena 5162 Arista 7060(D)X4-32

Bezpečnost	Bezpečnost jako služba Efektivní ochrana před DDOS Využití strojového učení pro analýzu útoků Řešení DDOS na úrovni ISP a SD WAN Obrana pohybujícími se cíli	Spojení více zařízení do jednoho celku (IDS, *„firewall“,* vyvažování zátěže…) Automatizace nasazení služeb a rekonfigurace Rychlé prototypování	Pica8 P-5401 Centec V580-48X2Q4Z Cisco 3408
SD-WAN	Řešení útoků na straně ISP Efektivní QoS na úrovni ISP Dohled a dodržování SLA Možnost poskytovat přidané a komplexnější služby	Globální přehled o síti Posun k otevřenému zdrojovému kódu a větší spolupráci firem Univerzálnost HW Odklon od dedikovaných zařízení	Noviflow WB-5168 Ciena 5162 Accton VED2181F
Mobilní sítě	Virtualizace jednotlivých služeb v MPC a jejich snadná migrace Řešení QoE (MOS a výběr cest) *„Edge computing“* Efektivní vytěžování služeb D2D komunikace	Abstrakce a globální přehled o sítí Neutralita výrobců a dodavatelů Rychlé prototypování Odklon od dedikovaných zařízení Rychlá migrace virtuálních strojů	Huawei AR3600 series Cisco 3408 Nuage Networks 210 WBX

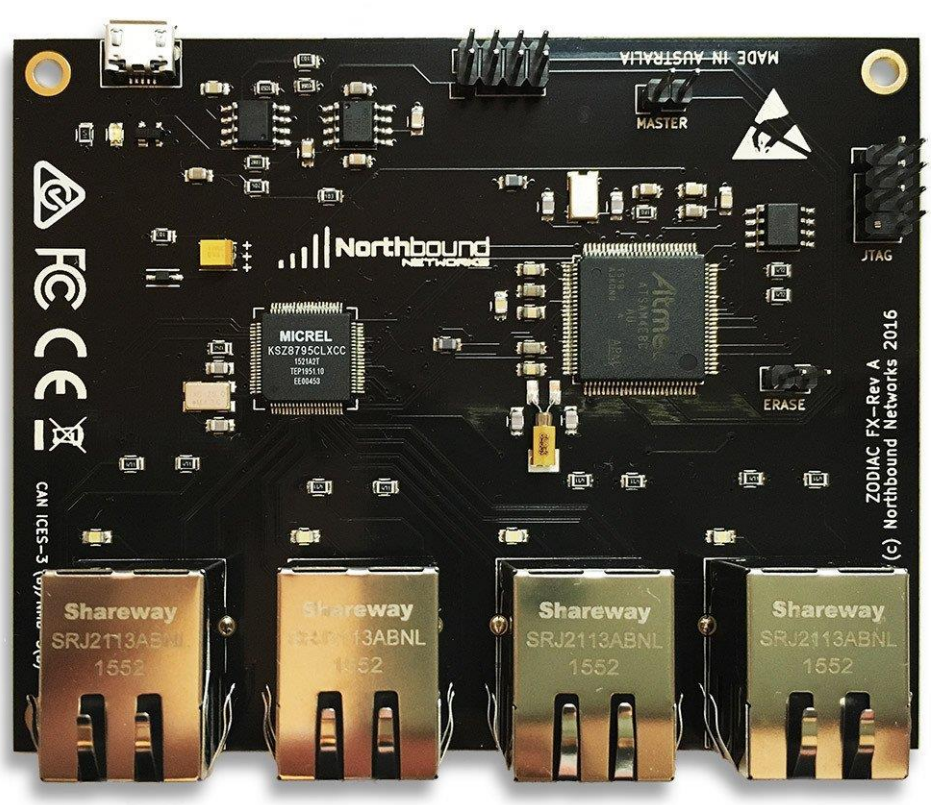
5 Návrh a stavba laboratorní platformy

5.1 Volba řešení a výběr komponent

Pro testování SDN a jejich aplikací bylo v minulosti vyvinuto mnoho nástrojů. Mezi ty nejznámější v kategorii *„open source“* patří například emulační platforma *Mininet*, či virtualizace za pomocí *Open vSwitch*.

Mininet umožňuje emulovat reálné sítě a jejich parametry v rámci jednoho zařízení. Díky dobré optimalizaci a rozložení jednotlivých síťových prvků na jednotlivé procesy dokáže *Mininet* fungovat i na průměrně výkonných PC. *Open vSwitch* je virtuální přepínač poskytující široké portfolio funkcionalit, rozhraní pro správu a nativní podporu *Openflow*. Mimo samotný *Mininet* jej podporuje či přímo využívá mnoho platforem, mezi které patří například *VirtualBox, XenServer, OpenNebula* aj. [68]

Pro naši laboratorní platformu byla zvažována možnost úplné virtualizace, například v uvedeném *Mininetu*, avšak hlavně kvůli potenciálnímu využití ve výuce, vyšší názornosti a využití většího spektra technologií bylo přikročeno k co největší hardwarové implementaci. Jakožto hlavní kandidát na pozici přepínače byl uvažován *ZodiacFX* od firmy *Northbound Networks*, který obsahuje 4 ethernet porty, síťovou čipovou sadu od *Microchip* a procesor *Atmel*. [69] Z důvodu nedostupnosti a dočasného pozastavení výroby v době zpracovávání možných řešení nebyl nakonec vybrán jako vhodné řešení.



Obrázek 5.1 - SDN přepínač *ZodiacFX* [69]

Díky řádově nižší pořizovací ceně, obecnému rozšíření, hojné uživatelské podpoře a snadné dostupnosti bylo nakonec jako hlavní zařízení zvoleno *RaspberryPi*, konkrétně nejvýkonnější 64bitová verze 3B+. *Raspberry* je možné dále rozšířit až o 4 *ethernetová* rozhraní za pomocí USB síťových karet, na celkový počet pěti rozhraní. Pro minimalizaci nákladů byl zvolen čínský adaptér, označovaný jako QTS1081, který je globálně dostupný například z *Ebay.com* přibližně za 1.5 dolaru. Ten slibuje podporu 100Mbps varianty *ethernetu* za pomocí USB2.0 rozhraní.

Za operační systém pro *RaspberryPi* byl zvolen osvědčený *Raspbian Stretch Lite* s verzí jádra 4.14.79. Jednotlivá *RaspberryPi* jsou dále nakonfigurována jako síťová zařízení za pomoci *Open vSwitche*. Za používaný kontrolér byl zvolen *Ryu*, který je díky relativně malé zátěži schopný běhu na platformě *RaspberryPi*. Správa všech zařízení je zajištěna pomocí protokolu SSH.

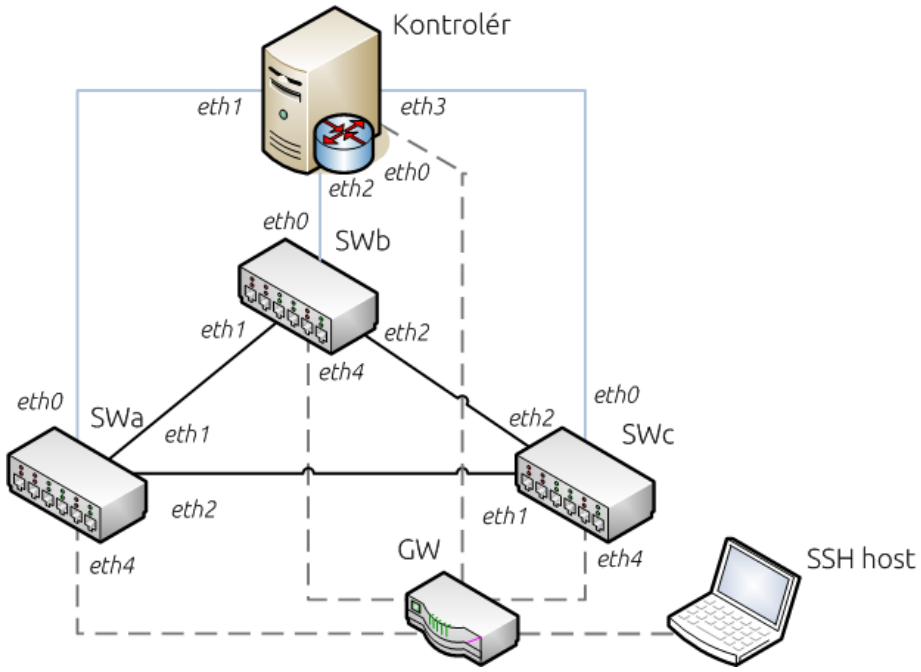
5.2 Topologie a adresování

Vzhledem k omezenému škálování daného počtu rozhraní byla po vyhrazení jednoho rozhraní pro správu zvolena kruhová topologie o třech přepínačích. V případě odebrání vyhrazeného rozhraní by bylo možné platformu rozšířit o čtvrtý síťový prvek. Pro pohodlné testování je však důležitý vzdálený přístup a dostupné připojení k internetu, proto je zde uvažována topologie se třemi prvky.

Tabulka 5.1 popisuje navržené síťové adresy. Rozhraní přepínačů *eth4* jsou vyhrazena pro SSH a nativní rozhraní (*eth0*) jsou používány pro komunikaci s kontrolérem. Rozhraní *eth1* – *eth3* budou přiřazena do síťového mostu, se kterým pracuje *Open vSwitch*. Obrázek 5.2 dále ilustruje zapojení celé platformy.

Tabulka 5.1 - Navrhované adresy pro laboratorní platformu

<table>
<tr><th colspan="2">Kontrolér</th><th colspan="2">SWa</th><th colspan="2">SWb</th><th colspan="2">SWc</th></tr>
<tr><th>Port</th><th>Adresa</th><th>Port</th><th>Adresa</th><th>Port</th><th>Adresa</th><th>Port</th><th>Adresa</th></tr>
<tr><td>eth0</td><td>192.168.0.10</td><td>eth0</td><td>192.168.1.11</td><td>eth0</td><td>192.168.2.11</td><td>eth0</td><td>192.168.3.11</td></tr>
<tr><td>eth1</td><td>192.168.1.10</td><td>eth1</td><td rowspan="3">Síťový most</td><td>eth1</td><td rowspan="3">Síťový most</td><td>eth1</td><td rowspan="3">Síťový most</td></tr>
<tr><td>eth2</td><td>192.168.2.10</td><td>eth2</td><td>eth2</td><td>eth2</td></tr>
<tr><td>eth3</td><td>192.168.3.10</td><td>eth3</td><td>eth3</td><td>eth3</td></tr>
<tr><td>eth4</td><td>192.168.4.10</td><td>eth4</td><td>192.168.0.11</td><td>eth4</td><td>192.168.0.12</td><td>eth4</td><td>192.168.0.13</td></tr>
<tr><td colspan="4">Podsíť LAN (správa a přístup k Internetu)</td><td colspan="4">192.168.0.0</td></tr>
<tr><td colspan="4">Brána</td><td colspan="4">192.168.0.1</td></tr>
<tr><td colspan="4">SSH klient</td><td colspan="4">192.168.0.250</td></tr>
<tr><td colspan="4">Podsíť pro připojená zařízení</td><td colspan="4">192.168.100.0</td></tr>
</table>



Obrázek 5.2 - Schéma zapojení laboratorní platformy

5.3 Zapojení a první start zařízení

Po prvním startu *Raspbianu* je vhodné změnit výchozí heslo a aktualizovat systém příkazy:

```
su
apt-get update
apt-get upgrade
passwd
```

Dále můžeme pro větší přehlednost změnit název zařízení z výchozího na *SWx* či *Controller*:

```
sudo nano /etc/hostname
sudo nano /etc/hosts
```

Vytvořením prázdného souboru *ssh* ve složce *boot se* SSH server automaticky spustí po *bootu* zařízení.

```
sudo nano /boot/ssh
```

Dále provedeme vypnutí DHCP služby příkazem:

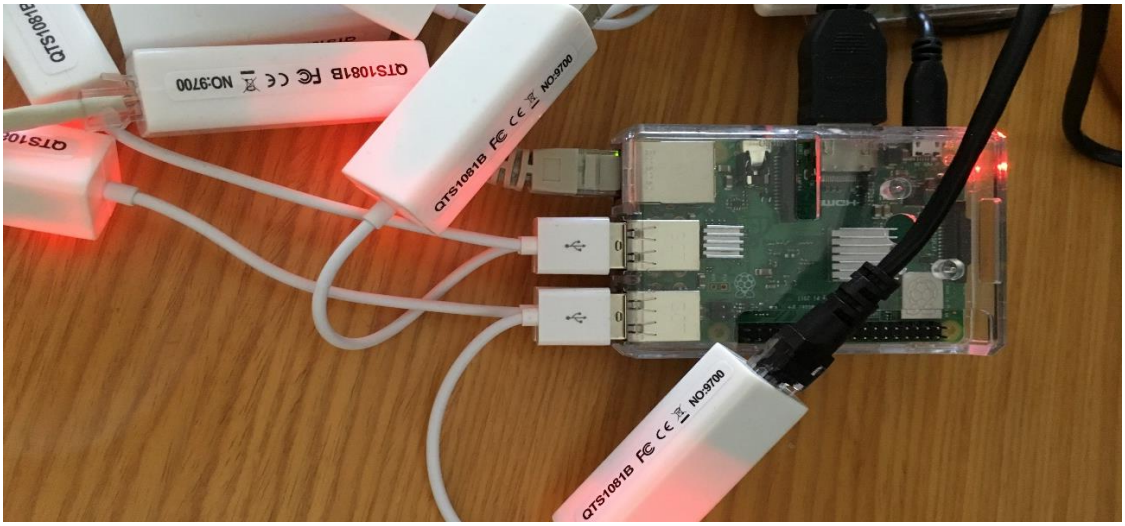
```
sudo update-rc.d -f dhcpcd remove
```

V případě potřeby jej lze opětovně zapnout příkazem:

```
sudo update-rc.d -f dhcpcd defaults
```

Po připojení síťových karet můžeme jejich úspěšné přidání do systému zkontrolovat pomocí příkazů:

`ifconfig` nebo `ip a`



Obrázek 5.3 - *RaspberryPi* s připojenými USB kartami

Při pohledu na výpis všech dostupných rozhraní je patrné, že tyto levné síťové karty mají všechny stejnou adresu MAC, což by neumožnilo protokolu ARP správně, či vůbec fungovat. Pro automatickou změnu adres i po restartu zařízení můžeme s výhodou využít pravidel ve složce *udev*. Pomocí následujícího příkazu vytvoříme *soubor.rules*, kdy předcházející číslo určuje prioritu před ostatními pravidly.

```
sudo nano /etc/udev/rules.d/70-persistent-net.rules
```

Vzhledem k nedostupnosti jednoznačných identifikátorů, jako jsou *IDvendor, IDproduct* či *serial number* (viz obrázek 5.4) jsou následující pravidla specifikována pouze podle atributů MAC adresa a název. Ty pomocí příkazu `ifconfig` přepíší MAC adresu daného rozhraní. Tyto změny proběhnou vždy při *bootu* zařízení.

```
pi@SWc:~ $ lsusb -v

Bus 001 Device 006: ID 0fe6:9700 Kontron (Industrial Computer Source / ICS Advent) DM9601 Fast Ethernet Adapter
Couldn't open device, some information will be missing
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               1.10
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x0fe6 Kontron (Industrial Computer Source / ICS Advent)
  idProduct          0x9700 DM9601 Fast Ethernet Adapter
  bcdDevice            1.01
  iManufacturer           0
  iProduct                2
  iSerial                 0
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           39
    bNumInterfaces          1
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0xa0
      (Bus Powered)
      Remote Wakeup
    MaxPower              120mA
```

Obrázek 5.4 - Detailní výpis atributů USB síťové karty

Nové MAC adresy byly definovány podle následujícího klíče. V poslední dvojici je zaznamenáno číslo rozhraní (1-4) a pořadové písmeno přepínače (a-c). Pro kontrolér bylo zvoleno označení dvěma stejnými číslicemi. Následující výpis ukazuje pravidla pro *Controller a SWa.*

```
Controller
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:e0:4c:53:44:58", NAME="eth1", RUN+="/sbin/ifconfig
eth1 hw ether de:ad:be:ef:f1:11"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:e0:4c:53:44:58", NAME="eth2", RUN+="/sbin/ifconfig
eth2 hw ether de:ad:be:ef:f2:22"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:e0:4c:53:44:58", NAME="eth3", RUN+="/sbin/ifconfig
eth3 hw ether de:ad:be:ef:f3:33"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:e0:4c:53:44:58", NAME="eth4", RUN+="/sbin/ifconfig
eth4 hw ether de:ad:be:ef:f4:44"

SWa
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:e0:4c:53:44:58", NAME="eth1", RUN+="/sbin/ifconfig
eth1 hw ether de:ad:be:ef:f1:1a"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:e0:4c:53:44:58", NAME="eth2", RUN+="/sbin/ifconfig
eth2 hw ether de:ad:be:ef:f2:2a"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:e0:4c:53:44:58", NAME="eth3", RUN+="/sbin/ifconfig
eth3 hw ether de:ad:be:ef:f3:3a"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:e0:4c:53:44:58", NAME="eth4", RUN+="/sbin/ifconfig
eth4 hw ether de:ad:be:ef:f4:4a"
```

Jako další problém se ukázalo na první pohled náhodné pořadí, ve kterém se připojené síťové karty přidávají do systému. Vycházíme z předpokladu, že při čelním pohledu na *RaspberryPi* číslujeme USB rozhraní od shora dolů a zleva doprava.

Obrázek 5.5 - Značení USB rozhraní

Výpis stromové struktury všech USB zařízení nám dává představu o větvení USB rozbočovače a cestu k připojeným zařízením.

```
pi@SWc:~ $ lsusb -t
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=dwc_otg/1p, 480M
    |__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M
        |__ Port 1: Dev 3, If 0, Class=Hub, Driver=hub/3p, 480M
            |__ Port 2: Dev 5, If 0, Class=(Defined at Interface level), Driver=dm9601, 12M
            |__ Port 3: Dev 7, If 0, Class=(Defined at Interface level), Driver=dm9601, 12M
            |__ Port 1: Dev 8, If 0, Class=Vendor Specific Class, Driver=lan78xx, 480M
        |__ Port 2: Dev 4, If 0, Class=(Defined at Interface level), Driver=dm9601, 12M
        |__ Port 3: Dev 6, If 0, Class=(Defined at Interface level), Driver=dm9601, 12M
pi@SWc:~ $
```

Obrázek 5.6 - Stromová struktura USB rozbočovače

Další informace je možné zjistit z *logu* jádra, který po startu systému přidání karet zaznamenává. Zde se dozvídáme mapování konkrétních ethernetových rozhraní na rozhraní USB rozbočovače. Pro vypsání informací o USB zařízeních můžeme použít například příkaz:

```
dmesg | grep USB
```

```
dm9601 1-1.2:1.0 eth1: register 'dm9601' at usb-3f980000.usb-1.2, Davicom DM96xx USB 10/100 Ethernet, 00:e0:4c:53:44:58
dm9601 1-1.1.2:1.0 eth2: register 'dm9601' at usb-3f980000.usb-1.1.2, Davicom DM96xx USB 10/100 Ethernet, 00:e0:4c:53:44:58
dm9601 1-1.3:1.0 eth3: register 'dm9601' at usb-3f980000.usb-1.3, Davicom DM96xx USB 10/100 Ethernet, 00:e0:4c:53:44:58
dm9601 1-1.1.3:1.0 eth4: register 'dm9601' at usb-3f980000.usb-1.1.3, Davicom DM96xx USB 10/100 Ethernet, 00:e0:4c:53:44:58
```

Obrázek 5.7 - Výpis „*dmesg*“ týkající se síťových karet

Pomocí těchto výpisů jsme vyjasnili následující mapování USB rozhraní na *ethernetová* rozhraní. Vzhledem k již dříve zmíněné absenci jednoznačných identifikátorů, je místo přemapování pomocí pravidel nejjednodušším řešením fyzicky přestavět pozici síťových karet v platformě.

Tabulka 5.2 - Mapování USB rozhraní na ethernetová rozhraní

USB rozhraní	Ethernetové rozhraní	Cesta v USB rozbočovači
USB1	eth2	1-1.1.2
USB2	eth4	1-1.1.3
USB3	eth3	1-1.3
USB4	eth1	1-1.2

5.4 Konfigurace přepínačů

Nejprve je nutné stáhnout *Open vSwitch*. Například jako již zkompilovaný balíček pomocí příkazu:

```
sudo apt-get install bridge-utils OpenvSwitch-switch OpenvSwitch-
common
```

Dalším krokem je konfigurace rozhraní na přepínačích. Následuje vzorová konfigurace v `etc/network/interfaces` pro SWa, která rozhraním přiřazuje adresy podle tabulky 5.1.

```
#SWa
allow-hotplug eth0
iface eth0 inet static
    address 192.168.1.11
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
    gateway 192.168.1.10
    dns-server 8.8.8.8

allow-hotplug eth1
iface eth1 inet manual

allow-hotplug eth2
iface eth2 inet manual

allow-hotplug eth3
iface eth3 inet manual

allow-hotplug eth4
iface eth4 inet static
    address 192.168.0.11
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
    dns-server 8.8.8.8

auto BRa
allow-ovs BRa
iface BRa inet manual
    ovs_type OVSBridge
    ovs_ports eth1 eth2 eth3
```

Poslední odstavec popisuje *Open vSwitch síťový most BRa*, který bude dostupný i po restartu zařízení a bude obsahovat porty *eth1-3*.

Samotný síťový most ale musí být nadefinován pomocí následujících příkazů. Program *ovs-vsctl* poskytuje uživateli rozhraní pro přístup ke konfigurační databázi přepínače. Po přiřazení požadovaných rozhraní je nutné dále přidat adresu kontroléru a příkazem `connection-mode=out-of-band` říci přepínači, že kontrolér bude komunikovat po vyhrazeném rozhraní, namísto vnitřních rozhraní přepínače.

```
sudo ovs-vsctl add-br BRa
sudo ovs-vsctl set bridge BRa protocols=Openflow10, Openflow13
sudo ifconfig BRa up
sudo ovs-vsctl add-port BRa eth1
sudo ovs-vsctl add-port BRa eth2
sudo ovs-vsctl add-port BRa eth3
sudo ovs-vsctl set-controller BRa tcp:192.168.1.10:6633
sudo ovs-vsctl set controller BRa connection-mode=out-of-band
```

Po přidání síťového mostu, je záhodné jeho správné přidání zkontrolovat příkazem `ifconfig` nebo lépe `sudo ovs-ofctl show`, který vypisuje současnou konfiguraci OF přepínače.

```
pi@SWb:~ $ sudo ovs-vsctl set-controller BRb tcp:192.168.2.10:6633
pi@SWb:~ $ sudo ovs-vsctl show
391d5203-5ca1-47f5-9d1a-311c8ba8f858
    Bridge BRb
        Controller "tcp:192.168.2.10:6633"
        Port "eth3"
            Interface "eth3"
        Port "eth1"
            Interface "eth1"
        Port "eth2"
            Interface "eth2"
        Port BRb
            Interface BRb
                type: internal
    ovs_version: "2.3.0"
pi@SWb:~ $
```

Obrázek 5.8 - Správná konfigurace *Open vSwitch* síťového mostu *BRb*

5.5 Konfigurace kontroléru

S ohledem na potřebný výpočetní výkon, programovací jazyk, širokou uživatelskou základnu a v neposlední řadě kvalitně zpracovanou dokumentaci byl vybrán kontrolér *Ryu*. Nejsnazší cestou k jeho získání je manažer *Python* balíčků PIP.

```
sudo apt-get install python-dev python3-pip
sudo pip3 install ryu
```

Dále nakonfigurujeme síťová rozhraní kontroléru v `etc/network/interfaces` podle tabulky 5.1.

```
auto eth0
iface eth0 inet static
    address 192.168.0.10
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
    dns-server 8.8.8.8
```

```
auto eth1
iface eth1 inet static
    address 192.168.1.10
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255

auto eth2
iface eth2 inet static
    address 192.168.2.10
    netmask 255.255.255.0
    network 192.168.2.0
    broadcast 192.168.2.255

auto eth3
iface eth3 inet static
    address 192.168.3.10
    netmask 255.255.255.0
    network 192.168.3.0
    broadcast 192.168.3.255

auto eth4
iface eth4 inet static
    address 192.168.4.10
    netmask 255.255.255.0
    network 192.168.4.0
    broadcast 192.168.4.255
```

V tuto chvíli by mělo být přístupné připojení přes SSH ke všem prvkům. Za jeho pomoci je možné ověřit konektivitu, například pomocí příkazu `ping` mezi jednotlivými přepínači a kontrolérem.

Obrázek 5.9 ukazuje výpis *logu Open vSwitche*, který se zpočátku pokoušel navázat spojení s kontrolérem bez definování podporované verze OF na 1.3. Po přidání OF verze 1.3 je možné vidět úspěšné navázání spojení.

```
pi@SWb:~ $ tail -f /var/log/openvswitch/ovs-vswitchd.log
2019-04-09T06:23:59.606Z|06959|rconn|INFO|BRb<->tcp:192.168.2.10:6633: connected
2019-04-09T06:23:59.833Z|06960|fail_open|WARN|No longer in fail-open mode
2019-04-09T06:24:09.834Z|06961|connmgr|INFO|BRb<->tcp:192.168.2.10:6633: 1 flow_mods 10 s ago (1 adds)
2019-04-09T06:34:23.639Z|06962|vconn|WARN|unix: version negotiation failed (we support version 0x04, peer suppo
rts version 0x01)
2019-04-09T06:34:23.639Z|06963|rconn|WARN|BRb<->unix: connection dropped (Protocol error)
2019-04-09T06:53:16.212Z|06964|connmgr|INFO|BRb: re-added service controller "punix:/var/run/openvswitch/BRb.mg
mt"
2019-04-09T06:53:16.213Z|06965|connmgr|INFO|BRb: re-added primary controller "tcp:192.168.2.10:6633"
2019-04-09T06:53:16.213Z|06966|rconn|INFO|BRb<->tcp:192.168.2.10:6633: connecting...
2019-04-09T06:53:16.228Z|06967|rconn|INFO|BRb<->tcp:192.168.2.10:6633: connected
2019-04-09T06:53:26.244Z|06968|connmgr|INFO|BRb<->tcp:192.168.2.10:6633: 1 flow_mods 10 s ago (1 adds)
```

Obrázek 5.9 - Výpis *„logu“ Open vSwitche* o návázání spojení s kontrolérem

Obrázek 5.10 ukazuje výchozí záznam *„flow“* pro *SWb*, který veškerý provoz směruje směrem na kontrolér.

```
pi@SWb:~ $ sudo ovs-ofctl dump-flows BRb
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=164.360s, table=0, n_packets=0, n_bytes=0, idle_age=1920,
priority=0 actions=CONTROLLER:65535
pi@SWb:~ $
```

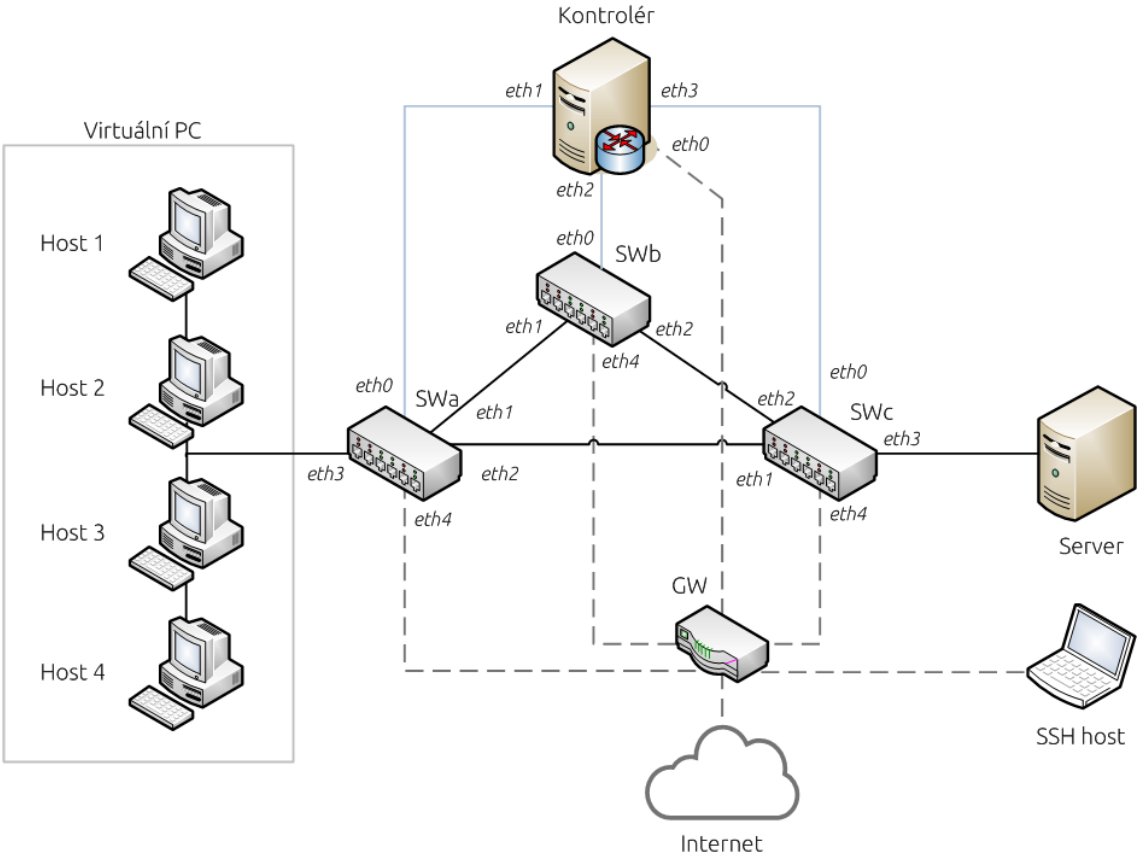
Obrázek 5.10 - Výchozí záznam „*flow*“ tabulky *SWb*

5.6 Ověření funkce

Pro ověření správné funkce celého testovacího stanoviště byla vybrána kruhová topologie o třech přepínačích, kdy na kontroléru poběží aplikace pro přepínače s podporou protokolu STP, který předchází vzniku síťových smyček.

Po propojení všech prvků podle schématu na obrázku 5.11 můžeme přikročit ke konfiguraci *hostů*. Virtuální *hosté* byli vytvořeni pomocí *VirtualBoxu* a *Linuxové* distribuce *Archlinux*. Zde je důležité ve *Virtualboxu* nastavit síťový most, který umožní virtuálním systémům komunikovat svojí adresou, na místo překladu na adresu hostujícího PC.

Připojeným zařízením byly přidány adresy z rozsahu *192.168.100. 101 – 192.168.100.104*. Hostující PC běžící na *Windows* 7 komunikuje na adrese *192.168.100.100*. Pro simulaci serveru bylo použito další *RaspberryPi s Raspbianem* a adresou *192.168.100.10*.



Obrázek 5.11 - Schéma zapojení pro ověření funkce

Balíček kontroléru *Ryu* v sobě obsahuje několik základních SDN aplikací a rozšíření. Jednou z nich je také zmíněný přepínač podporující STP. Pro jeho spuštění využijeme následující příkaz s parametrem *verbose* pro plný výpis.

```
ryu-manager --verbose ryu.app.simple_monitor_13
```

Následuje relativně dlouhý výpis domluvy STP přepínačů, které na základě nejnižší hodnoty identifikátoru vybírají kořenový přepínač a vybírají cesty s nejmenšími náklady. Nakonec zablokují rozhraní, na kterých by vznikala síťová smyčka. Obrázek 5.12 ukazuje část výpisu o zablokování portu *eth2* na *SWc*, tedy spojení mezi *SWa* a *SWc* přes *SWb*.

```
[STP][INFO] dpid=000000e04c534458: [port=1] ROOT_PORT           / LISTEN
[STP][INFO] dpid=000000e04c534458: [port=2] NON_DESIGNATED_PORT / LISTEN
[STP][INFO] dpid=000000e04c534458: [port=3] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=000076d2bf1ec04b: [port=1] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=000076d2bf1ec04b: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=000076d2bf1ec04b: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000dac7a9cf6740: [port=1] ROOT_PORT           / LEARN
[STP][INFO] dpid=0000dac7a9cf6740: [port=2] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=0000dac7a9cf6740: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=000000e04c534458: [port=1] ROOT_PORT           / LEARN
[STP][INFO] dpid=000000e04c534458: [port=2] NON_DESIGNATED_PORT / LEARN
[STP][INFO] dpid=000000e04c534458: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=000076d2bf1ec04b: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=000076d2bf1ec04b: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=000076d2bf1ec04b: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000dac7a9cf6740: [port=1] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000dac7a9cf6740: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000dac7a9cf6740: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=000000e04c534458: [port=1] ROOT_PORT           / FORWARD
[STP][INFO] dpid=000000e04c534458: [port=2] NON_DESIGNATED_PORT / BLOCK
[STP][INFO] dpid=000000e04c534458: [port=3] DESIGNATED_PORT     / FORWARD
```

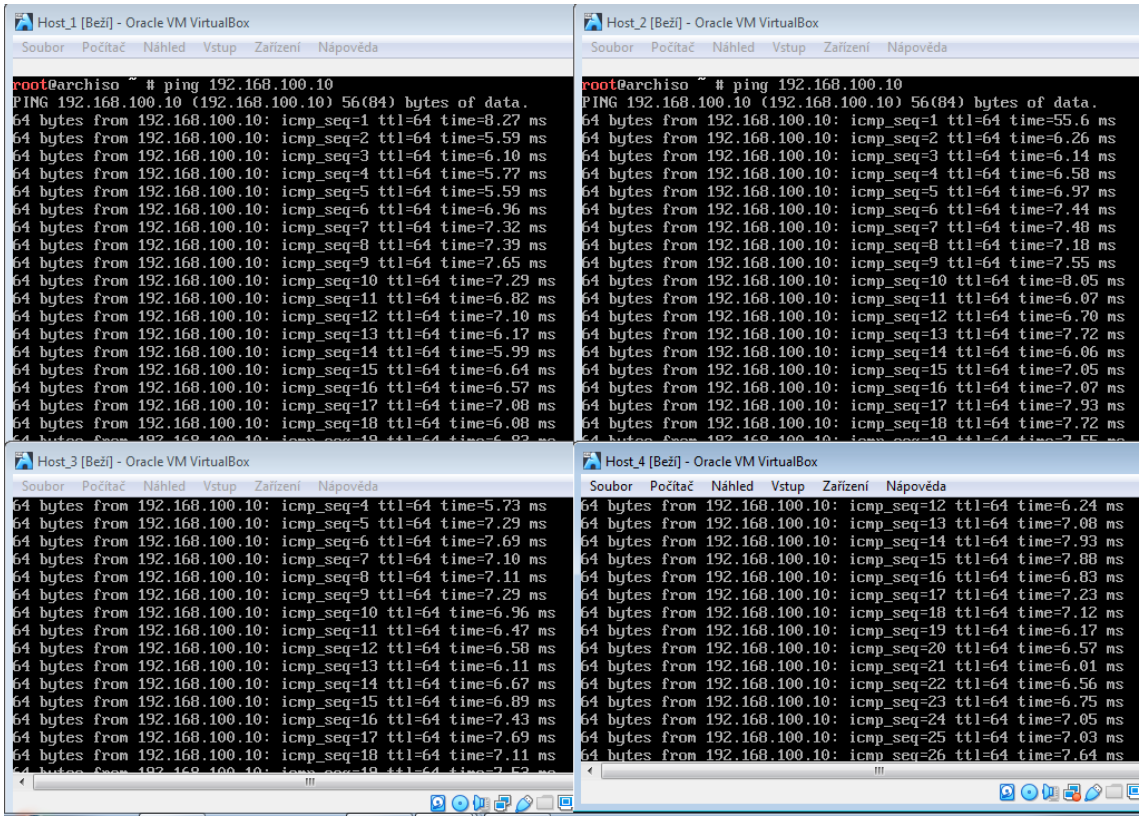
Obrázek 5.12 - Výpis inicializace STP protokolu

Ověření vzájemné konektivity hostů a serveru proběhlo pomocí příkazu ping ze čtyř virtuálních *hostů* směrem na server. Obrázek 5.13 ukazuje oříznutý výpis „*flow*“ při testu konektivity.

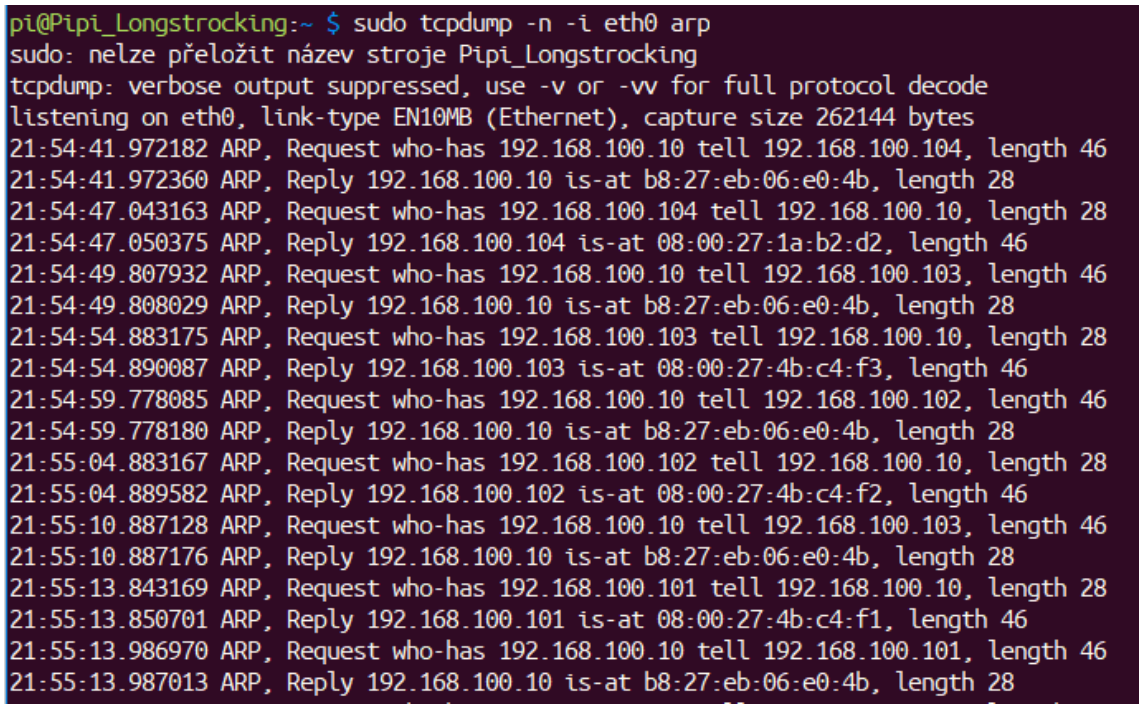
```
packet in 963353199704 2c:d4:44:af:7f:e1 ff:ff:ff:ff:ff:ff 1
packet in 240551082288960 2c:d4:44:af:7f:e1 ff:ff:ff:ff:ff:ff 1
packet in 963353199704 b8:27:eb:06:e0:4b 2c:d4:44:af:7f:e1 3
packet in 130647521673291 b8:27:eb:06:e0:4b 2c:d4:44:af:7f:e1 2
packet in 130647521673291 2c:d4:44:af:7f:e1 b8:27:eb:06:e0:4b 3
packet in 963353199704 2c:d4:44:af:7f:e1 b8:27:eb:06:e0:4b 1
packet in 130647521673291 2c:d4:44:af:7f:e1 33:33:00:01:00:02 3
packet in 963353199704 2c:d4:44:af:7f:e1 33:33:00:01:00:02 1
packet in 240551082288960 2c:d4:44:af:7f:e1 33:33:00:01:00:02 1
packet in 130647521673291 2c:d4:44:af:7f:e1 33:33:00:01:00:02 3
packet in 240551082288960 2c:d4:44:af:7f:e1 33:33:00:01:00:02 1
packet in 963353199704 2c:d4:44:af:7f:e1 33:33:00:01:00:02 1
```

Obrázek 5.13 - Výpis „*flow*“ z kontroléru při testu konektivity

Obrázek 5.14 ukazuje úspěšný ping z virtuálních PC na server. Obrázek 5.15 nabízí náhled na ARP komunikaci zachycenou na serveru.



Obrázek 5.14 – *„Ping“* z virtuálních PC směrem na server



Obrázek 5.15 - Záchyt ARP komunikace mezi zařízeními

Funkčnost STP protokolu byla dále ověřena fyzickým odpojením portu *eth1* na *SWa*, přičemž byl očekávaný pád přímého spojení mezi *SWa* a *SWc* a pokračování další komunikace přes horní *SWb*. Tuto skutečnost potvrzují obrázek 5.16 a obrázek 5.17.

```
[STP][INFO] dpid=000076d2bf1ec04b: [port=2] Link down.
[STP][INFO] dpid=000076d2bf1ec04b: [port=2] DESIGNATED_PORT     / DISABLE
[STP][INFO] dpid=000000e04c534458: [port=1] Link down.
[STP][INFO] dpid=000000e04c534458: [port=1] DESIGNATED_PORT     / DISABLE
[STP][INFO] dpid=000000e04c534458: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=000000e04c534458: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=000000e04c534458: Root bridge.
[STP][INFO] dpid=000000e04c534458: [port=2] DESIGNATED_PORT     / LISTEN
[STP][INFO] dpid=000000e04c534458: [port=3] DESIGNATED_PORT     / LISTEN
packet in 130647521673291 2c:d4:44:af:7f:e1 ff:ff:ff:ff:ff:ff 3
packet in 240551082288960 2c:d4:44:af:7f:e1 ff:ff:ff:ff:ff:ff 1
[STP][INFO] dpid=000000e04c534458: [port=2] Receive superior BPDU.
[STP][INFO] dpid=000000e04c534458: [port=2] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=000000e04c534458: [port=3] DESIGNATED_PORT     / BLOCK
[STP][INFO] dpid=000000e04c534458: Non root bridge.
[STP][INFO] dpid=000000e04c534458: [port=2] ROOT_PORT           / LISTEN
[STP][INFO] dpid=000000e04c534458: [port=3] DESIGNATED_PORT     / LISTEN
```

Obrázek 5.16 - Pád spojení mezi *SWa* a *SWc*

```
[STP][INFO] dpid=000000e04c534458: [port=2] ROOT_PORT           / LEARN
[STP][INFO] dpid=000000e04c534458: [port=3] DESIGNATED_PORT     / LEARN
[STP][INFO] dpid=000000e04c534458: [port=2] ROOT_PORT           / FORWARD
[STP][INFO] dpid=000000e04c534458: [port=3] DESIGNATED_PORT     / FORWARD
```

Obrázek 5.17 - Reakce *SWc* na pád spojení mezi *SWa* a *SWc*

Jako doplňkovou aplikaci pro zjištění statistik provozu lze využít *simple_monitor_13.py*, která periodicky vypisuje celkové statistiky přepínačů od spuštění a aktuální provoz. Pro jeho spuštění v kombinaci s STP přepínačem je nutné v jeho zdrojovém kódu (viz obrázek 5.18) importovat *simple_switch_stp_13* a přidat *SimpleMonitor13* jako dědice třídy *simple_switch_stp_13*. Obrázek 5.19 ukazuje souhrnný výpis veškerého provozu od doby startu přepínačů a statistiku pro současně probíhající „*flow*“.

```
 GNU nano 2.7.4                Soubor: /usr/local/lib/python3.5/dist-packages/ryu/app/simple_monitor_13.py


from ryu.app import simple_switch_stp_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub


class SimpleMonitor13(simple_switch_stp_13.SimpleSwitch13):

    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)
```

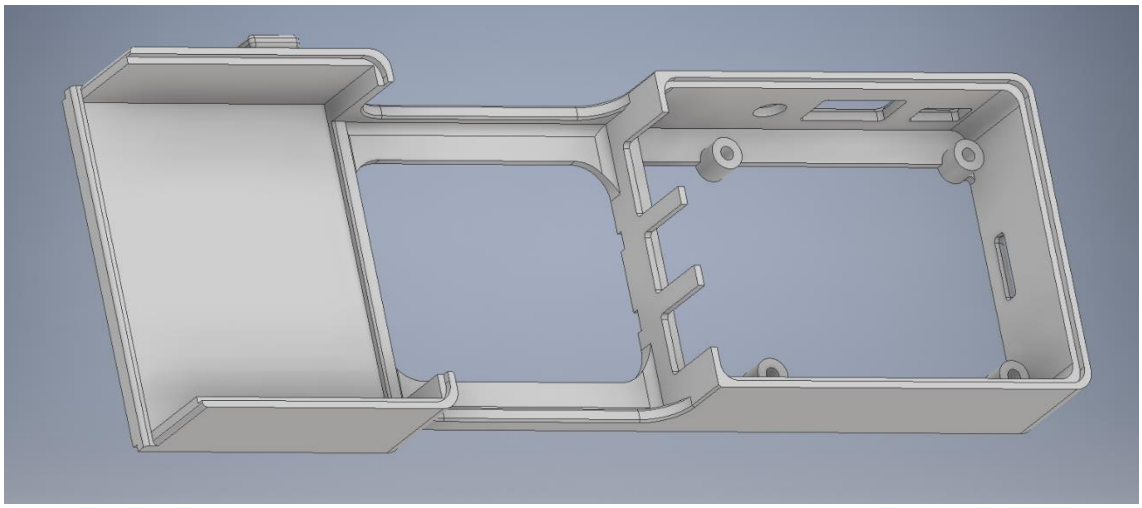
Obrázek 5.18 - Změna kódu aplikace *simple_monitor_13*

```
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
000076d2bf1ec04b        1 2c:d4:44:af:7f:e1        3       24     2276
000076d2bf1ec04b        3 b8:27:eb:06:e0:4b        1       23     2216
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
000076d2bf1ec04b        1    67086 13917472        0   691981 75035628      170
000076d2bf1ec04b        2   675496 63159041        6    75715 15657790      170
000076d2bf1ec04b        3    20096  1161036     1908   568190 61559154        0
000076d2bf1ec04b fffffffe   753407 77565440        0      277    21298        0
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
000000e04c534458        2 b8:27:eb:06:e0:4b        3       20     1922
000000e04c534458        3 2c:d4:44:af:7f:e1        2       20     1922
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
000000e04c534458        1    74829 14390537        0   674852 73902562        0
000000e04c534458        2   689317 63792345        0    65232 14858152        0
000000e04c534458        3     3067   247065        0   790597 91777992        0
000000e04c534458 fffffffe   750752 77413240        0      229    17938        0
datapath         in-port  eth-dst           out-port packets  bytes
---------------- -------- ----------------- -------- -------- --------
0000dac7a9cf6740        1 b8:27:eb:06:e0:4b        2       20     1922
0000dac7a9cf6740        2 2c:d4:44:af:7f:e1        1       20     1922
datapath         port     rx-pkts  rx-bytes rx-error tx-pkts  tx-bytes tx-error
---------------- -------- -------- -------- -------- -------- -------- --------
0000dac7a9cf6740        1   691090 63895810        8    66156 14917800        0
0000dac7a9cf6740        2    65224 13813460        0   689422 74830206        0
0000dac7a9cf6740        3        0        0        0        0        0        0
0000dac7a9cf6740 fffffffe   750576 77405580        0      255    18845        0
```

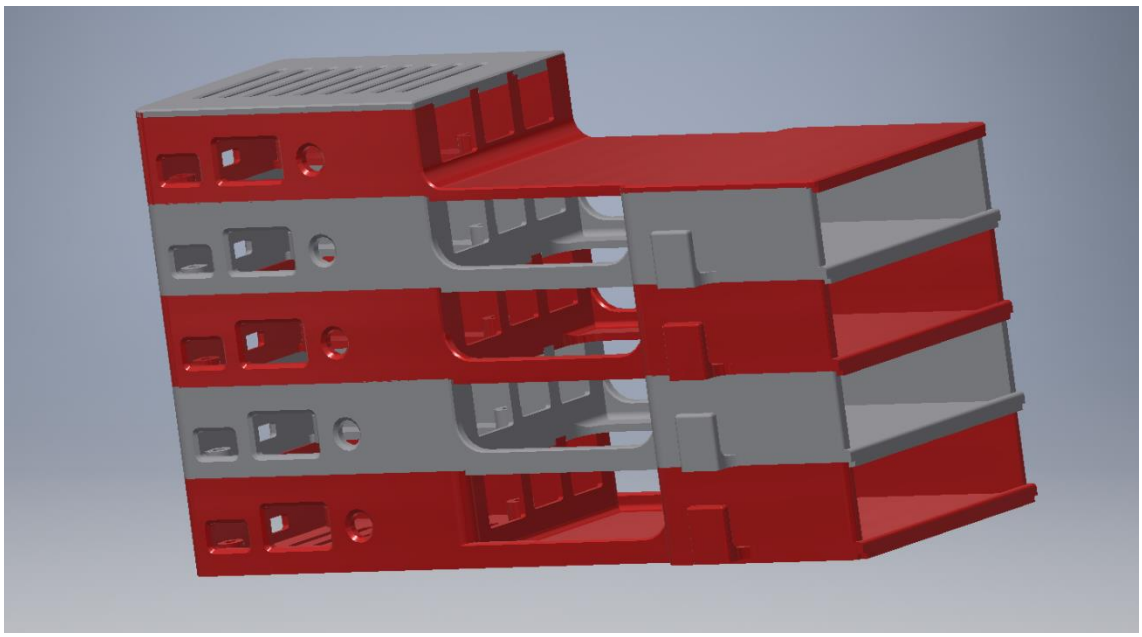
Obrázek 5.19 - Výpis aplikace *simple_monitor_13*

5.7 Stavba platformy

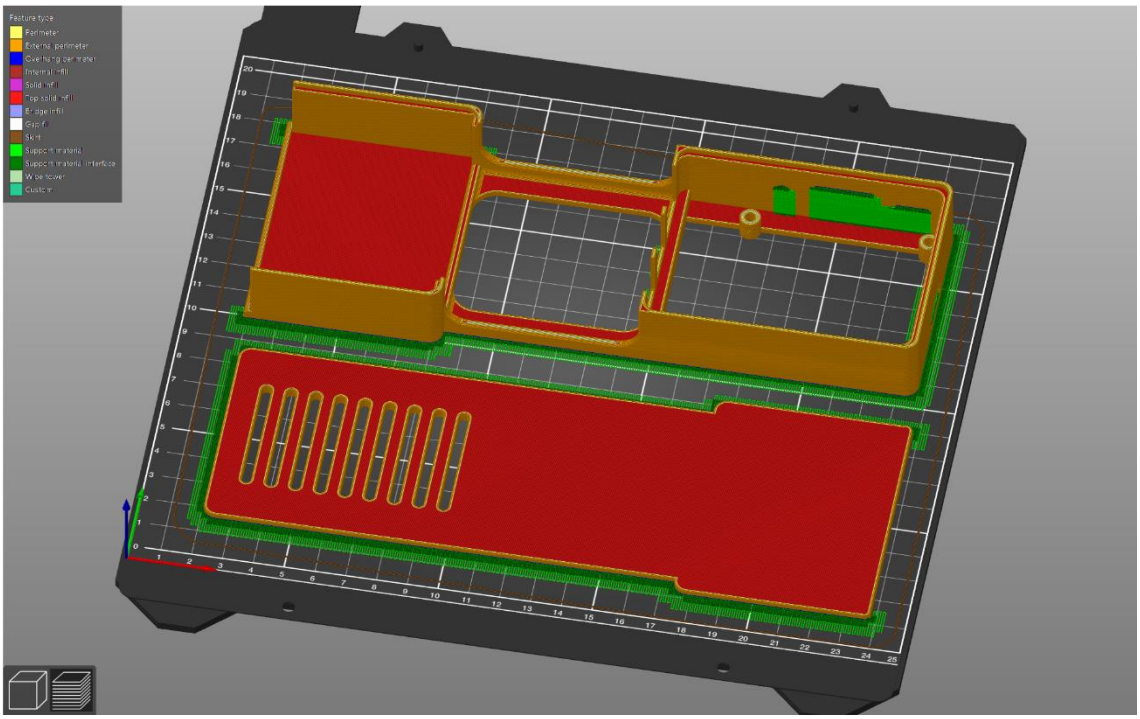
Pro uložení jednotlivých zařízení bylo navrhnuto následující plastové pouzdro, díky kterému je možné celou testovací stanici stohovat na sebe. Jednotlivé díly byly poté vytištěny na 3D tiskárně z plastu typu PLA. Díky tomuto pouzdru je celá platforma modulární, kompaktní a snadno přenositelná.



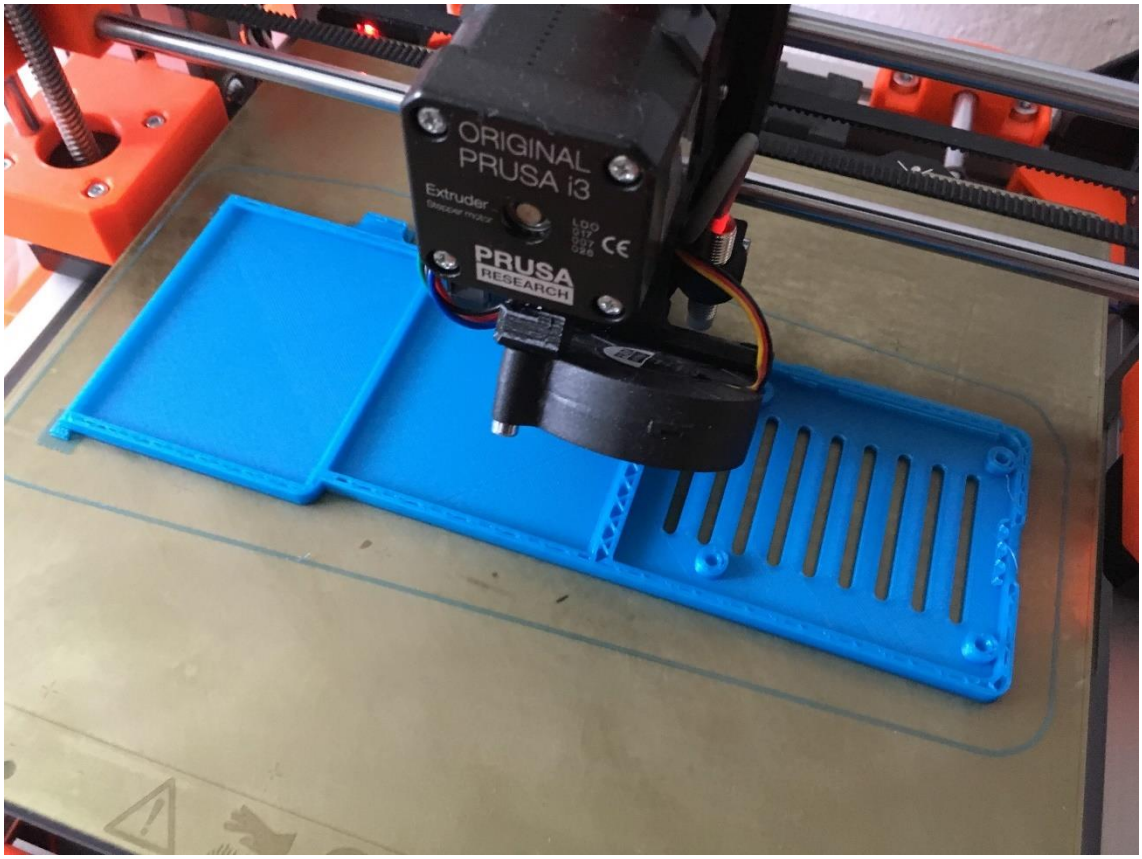
Obrázek 5.20 - Návrh plastového pouzdra



Obrázek 5.21 - Návrh sestavy celého testovacího stohu

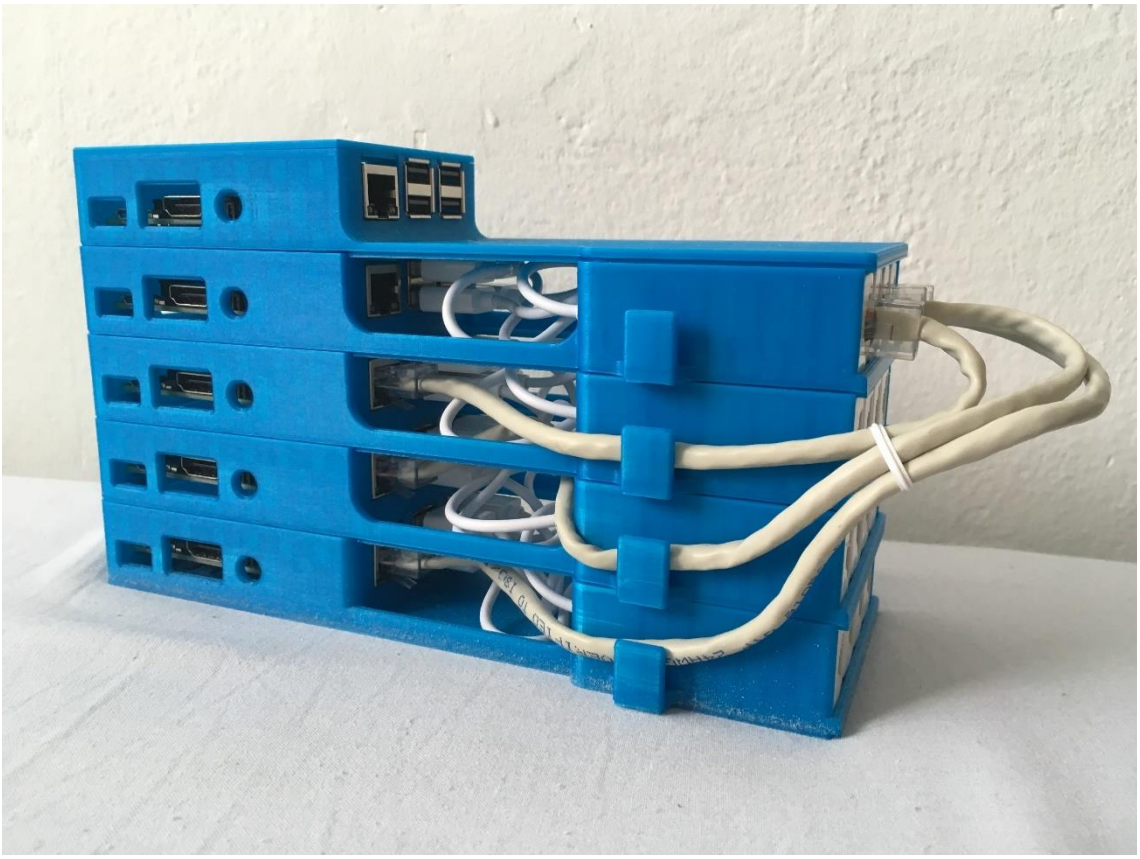


Obrázek 5.22 - Náhled dílu před tiskem v programu *Prusa Slic3r*



Obrázek 5.23 - Tisk spodního dílu sestavy

Obrázek 5.24 - Prostřední díl osazený *RaspberryPi*



Obrázek 5.25 - Kompletní testovací pracoviště

6 Vzorové aplikace pro výuku SDN

Tato kapitola byla zpracována jako podkladový materiál laboratorních cvičení pro studenty věnující se problematice SDN. Každá vypracovaná úloha obsahuje verzi pro studenta, podle které by měl být schopen úlohu samostatně dokončit. Dále pak verzi pro vyučujícího, která shrnuje potřebnou přípravu, časovou náročnost, či možné problémy při vypracování úloh. Vypracované úlohy jsou přiloženy v příloze práce. Texty pro úvod do problematiky byly převážně převzaty z předchozích kapitol, proto nejsou pro přehlednost v textu znovu citovány.

Závěr

Od prvních zmínek a experimentů s konceptem SDN již uběhlo několik let, a přesto se může na první pohled zdát, že se jedná pouze o zajímavou alternativu ke klasickým sítím, či pouhý akademický směr. Ačkoliv technologií pracujících v souladu s filozofií SDN neustále přibývá, globální nástup není takový, jak se odbornou veřejností předpokládalo. Při bližším pohledu na současný stav sítí a směr, kterým se vyvíjí informační technologie, se však jedná o přirozený a logický vývojový krok, který je čím dál tím více protlačován potřebami praxe.

Softwarově definované sítě se staly jakýmsi průkopníkem ve světě sítí, kdy můžeme pozorovat vývoj a často i hotová řešení předbíhající samotnou standardizaci protokolů či doporučení od standardizačních autorit. To sice umožňuje tvořit standardy na základě praxí poptávaných a ověřených řešeních, avšak také může vést k nedůvěře velkých firem investovat do nových technologií, které se mohou krátce po rozsáhlých investicích do infrastruktury vyvinout jiným směrem. Potřeby rozsáhlejší kontroly, programovatelnosti či rychlého prototypování vede velké společnosti, jako jsou FB, AWS AT&T aj. k tvorbě vlastních řešení, a tím zadávají podklady pro kvalitní a ověřená vzorová řešení a přirozeně vyvíjí tlak na usměrnění dalšího vývoje sítí směrem k SDN.

Jednou z prvotních myšlenek SDN byla otevřenost a odklonění se od uzavřenosti protokolů, platforem a podpory konkrétních výrobců. Nedostatečná standardizace a snaha o udržení místa na trhu však současně vede tradiční firmy z oboru telekomunikací jako je například Cisco k vzniku mnoha proprietárních řešení, což znovu udává směr k uzavřenosti celého SDN.

Ačkoliv se investice do SDN může pro investora v konzervativním prostředí telekomunikací zdát riskantní, přináší mnoho nezanedbatelných výhod. Mezi ně patří přímá ovlivnitelnost řídicí roviny, která umožní snazší prioritizaci, blokování služeb a uživatelů či vyvažování zátěže v „*multi-tenant cloud*“ architekturách. Stává se nezbytným pro vývoj a testování nových protokolů a významně urychluje nasazování nových aplikací a přístupů ve vysokorychlostních sítích. Umožňuje globální přehled, monitorování a ovlivnitelnost z jednoho místa, kdy pro konfiguraci stovek prvků s různými OS a rozhraními stačí pouze přístup ke kontroléru. Přináší snahu o abstrakci v podobě rozdělení funkčních rovin a nových jazyků, čímž umožňuje snazší programování aplikací. Dále tak mohou vznikat komplexnější aplikace, které na sebe mohou efektivněji navazovat, spolupracovat spolu a využívat stejných dostupných informací. Dále usnadňuje migraci služeb a zařízení, kdy nemusí docházet ke zdlouhavé rekonfiguraci *„firewallů“*, *„access* listů“ aj.

SDN také přináší větší volnost v možnosti volby dodavatelů a výrobců jednotlivých zařízení, aplikací a služeb. Díky tomu je možné jednotlivé síťové prvky a jejich OS v závislosti na podporovaných funkcionalitách téměř libovolně kombinovat a je tak potlačena závislost na jednotlivém výrobci. Neustálé zvyšování abstrakce, virtualizace a nárůstu výpočetního výkonu umožňuje virtualizaci některých drahých ASIC hardwarových řešení a tím snižování provozních i pořizovacích nákladů. SDN se tak stává nezbytným prvkem k efektivnímu provozování a řízení virtualizovaných služeb, sítí a hardware.

V případě, kdy dochází k centralizaci řídicí roviny, je otázka zabezpečení celého systému citlivější než kdy jindy. Musí být důsledně definována přístupová práva uživatelů i aplikací, jejich jednoznačná autentizace a omezeny možnosti útoků jako jsou například „*man in the middle*“, DDOS, dále úniky dat či odhalení směrovacích pravidel sledováním statistik či časování jednotlivých operací. Mezi další možné slabiny patří „*hijacking*“ kontroléru, vkládání padělaných pravidel a v neposlední řadě útok na jednotlivé prvky a slabiny v jejich OS.

Mezi další nevýhody SDN patří jednoznačně potřeba pořízení nového hardwaru, který podporuje *Openflow*. Tento protokol prošel několika verzemi, přičemž vyšší verze protokolu zpravidla přináší rozšíření funkcionalit a podporovaných polí. Tímto postupem se však stává vcelku objemným a objevují se tak diskuze o jeho nahrazení, což vnáší do nákupu drahého HW další pochybnosti.

Jakožto nově se vyvíjející fenomén provází SDN určitá nejasnost a nejistý směr, ke kterému nepřispívají často chybějící nebo nejednoznačně definované standardy či principy, na které jsme v oboru telekomunikací zvyklí. Další vývoj, ověřené praktiky a technologie, tak jako v případě každé nové technologie, ukáže až samotný čas.

SDN není klíčem ke globálnímu řešením všech problémů a před jeho implementací by měla předcházet citlivá rozvaha současného stavu a dalšího očekávaného vývoje dané sítě. Dnes se mezi nejmasovější místa nasazení řadí v drtivé většině „*cloud*“ a výpočetní centra s těsným zástupem sítí WAN a sítí telekomunikačních operátorů. Jakožto neustále se vyvíjející odvětví, v kterém vznikají mnohá více či méně otevřená řešení mající mnoho odnoží a variant, může SDN při prvním kontaktu působit jako velmi nepřehledná a komplexní problematika. Tato práce se tak mimo jiné snaží o popsání současného stavu, kategorizuje nejčastěji řešené problémy a nastiňuje jejich možné řešení pomocí SDN.

Praktická část přináší snadno dostupné a levné testovací pracoviště, které je mobilní a může tak s výhodami sloužit pro prezentaci SDN mimo prostředí laboratoří a serveroven. Pro školní potřeby je velmi názorné a jeho samotné sestavení mohou studenti provést během cvičení věnovaných výuce SDN či NFV. Dále oproti *Mininetu* ukazuje, že SDN je možné provozovat a testovat i na univerzálním a málo výkonném zařízení. Díky HW realizaci umožňuje snadné použití reálného provozu pro testování. Ačkoliv nebyla vzhledem ke zvolené platformě a omezenému počtu portů propustnost, či škálovatelnost hlavním měřítkem, je záhodné uvést jasné výkonnostní omezení v podobě přítomnosti pouze jednoho hlavního USB rozbočovače a zjištěné skutečnosti, že levné čínské karty ve skutečnosti využívají pouze standardu USB 1.1, což jejich reálnou propustnost omezuje na zhruba 12 Mbit/s.

Uvedené vzorové úlohy mají za úkol studenty seznámit se základní prací se systémem *Linux*, programovacím jazykem *Python*, principy fungování SDN a jeho výhodného využití pro řešení síťových služeb. Mezi další možná rozšíření platformy patří vytvoření bohatší topologie ať už přidáním čtvrtého přepínače, či využitím emulace v *Mininetu*. Dalším zajímavým aspektem by bylo provést měření propustnosti a výkonnosti celé platformy. Vypracované úlohy je možné rozšířit o více uživatelů a komplexnější algoritmus výběru v případě úlohy vyvažování zátěže, či testování reálného provozu s přesněji definovanými pravidly v případě úlohy integrace IDS.

Citovaná literatura

[1] What is Software Defined Networking (SDN). SDxCentral. [Online]. [cit. 2018-11-4]. Dostupné z: https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/.

[2] Koponen, Teemu. *Software is the future of networking.* Austin, Texas: ACM, 2012. Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems ISBN 978-1-4503-1685-9.

[3] Dungay, David. Software Defined Networking (SDN) Explained. CommsBusiness. [Online]. 23. 5. 2016 [cit. 2018-11-4]. Dostupné z: https://www.commsbusiness.co.uk/wp-content/uploads/2016/05/traditional-software-defined.png.

[4] Tittel, Ed. SDN vs. NFV: What's the difference? Cisco Systems Inc. [Online]. [cit. 2018-11-5]. Dostupné z: https://www.cisco.com/c/en/us/solutions/software-defined-networking/sdn-vs-nfv.html.

[5] SDN: The New Norm for Networks. *Open Networking.* [Online]. 13. 4. 2012 [cit. 2018-11-13]. Dostupné z: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[6] Charney, Neil. A Modern Platform for All Applications. *Docker.* [Online]. [cit. 2018-11-13]. Dostupné z: https://www.docker.com/why-docker.

[7] Carvalho, Larry, a další. IDC FutureScape: Worldwide Cloud 2018 Predictions. *IDC.* [Online]. 12. 2017 [cit. 2018-11-13]. Dostupné z: https://www.idc.com/getdoc.jsp?containerId=US43253017.

[8] Maillé, Patrick a Tuffin, Bruno. *Impact of Content Delivery Networks on service and and content innovation.* New York, USA : ACM, 12. 2015, Performance Evaluation Review, s. 49-52. DOI 10.1145/2847220.2847236.

[9] Feamster, Nick, Rexford, Jennifer a Zegura, Ellen. The road to SDN: an intellectual history of programmable networks. NY, USA: ACM, 4. 2014. *SIGCOMM Computer Communication Review.* vol. 44, s. 87 - 98.

[10] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov. Linux Netlink as an IP Services Protocol. *Internet Engineering Task Force.* 7. 2003, RFC 3549.

[11] L. Yang, R. Dantu, T. Anderson, and R. Gopal. Forwarding and Control Element Separation (ForCES) Framework. 4. 2004. *RFC 3746.*

[12] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo. The SoftRouter Architecture. San Diego, USA : ACM HOTNETS, 2004. 3nd ACM Workshop on Hot Topics in Networks. DOI 10.1.1.468.5468

[13] Igor Godanj, Krešimir Nenadić, Krešimir Romić. Simple Example of Software Defined Network. *Osijek, Croatia: IEEE,* 2016. International Conference on Smart Systems and Technologies (SST). ISBN 978-1-5090-3718-6

[14] Hong Yan, David A. Maltz, T. S. Eugene Ng, Hemant Gogineni. Tesseract: A 4D Network Control Plane. Cambridge, USA : NSDI, 2007. s. 27. 4th Symposium on Networked Systems Design and Implementation.

[15] Software Defined Everything Part 6: Infrastructure Attributes. *SDX Central.* [Online]. [cit. 2018-11-19]. Dostupné z: https://www.sdxcentral.com/cloud/definitions/software-defined-everything-part-6-infrastructure-attributes/.

[16] Kreutz, Diego, a další. *Software-Defined Networking: A Comprehensive Survey.* 2015. Proceedings of the IEEE . vol. 103. ISSN 1558-2256.

[17] Ghalwash, Haitham a Huang, Chun-Hsi. *A QoS framework for SDN-based Networks.* Philadelphia, PA, USA : IEEE, 2018. IEEE 4th International Conference on Collaboration and Internet Computing (CIC). ISBN 978-1-5386-9502-9.

[18] Braun, Wolfgang a Menth, Michael. *Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices.* Tuebingen, DE : MDPI AG, 2014. Future Internet. ISSN 1999-5903.

[19]*OpenFlow Switch Specification version 1.5.1.* ,Open Networking Foundation. 2015. [cit. 2018-11-19] Dostupné z: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[20] Seetharaman, Srini. *OpenFlow / Software-defined Networking.* Mountain View, USA: Deutsche Telekom Innovation Center, 2013. Beginners OpenFlow Tutorial Meetup. Dostupné z: https://www.google.cz/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&cad=rja&uact=8&ved=2ahUKEwiTgaPW_I7iAhUixcQBHZAVD30QFjAHegQICBAC&url=http%3A%2F%2Fyuba.stanford.edu%2F~srini%2FBeginners_OpenFlowTutorial_Meetup_June_2013.pptx&usg=AOvVaw286VDOxwXxh22XN1fdXTBR

[21] R. Wang, D. Butnariu, and J. Rexford. *OpenFlow-based server load balancing gone wild.* Boston, USA : USENIX, 2011. Proceedings of the USENIX Conference on Hot Topics in Management of Internet, Cloud, Enterprise Networks. s12.

[22] Jon, Matias, a další. *FlowNAC: Flow-based Network Access Control.* Bilbao, Spain: ETSI Bilbao, 2014. Third European Workshop on Software Defined Networks. ISSN 2379-0350

[23] Yakasai, Sadiq a Guy, Chris. *FlowIdentity: Software-Defined Network Access Control.* San Fracisco, USA: IEEE, 2015. IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN). ISBN 978-1-4673-6884-1.

[24] Brossard, David. XACML architecture and a sample authorization flow. *Wikipedia.* [Online]. 2016. [cit. 2018-11-20]. Dostupné z: https://en.wikipedia.org/wiki/XACML#/media/File:XACML_Architecture_%26_Flow.png.

[25] Amigo, Isabel, a další. *An SDN-Based Approach for QoS and Reliability in Overlay Networks.* Vienna, Austria: IEEE, 2018. Network Traffic Measurement and Analysis Conference . ISBN 978-1-5386-7152-8.

[26] What's the Difference Between Hybrid WAN and SD-WAN. *SDX Central.* [Online] [cit. 2018-11-22]. Dostupné z: https://www.sdxcentral.com/sd-wan/definitions/hybrid-wan-vs-sd-wan/.

[27] Rohit Mehra, Rajesh Ghai, and Brad Casemore. *SD-WAN: Momentum Builds as Early Adopters Experience Tangible Benefits.* Framingham, USA : IDC, 2018. IDC Technology Spotlight. Dostupné z: https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/intelligent-wan/idc-tangible-benefits.pdf

[28] Dufour, Maxime, a další. *Online Bandwidth Calendaring: On-the-fly admission, scheduling, and path computation.* Paris, France : IEEE, 2017. IEEE International Conference on Communications (ICC). ISBN 978-1-4673-9000-2.

[29] Jin, Feifei, a další. *Dynamic bandwidth management based on classification of traffic in software-defined WDM-TDM PON architecture.* Hangzhou, China : IEEE, 2016. 15th International Conference on Optical Communications and Networks (ICOCN). ISBN 978-1-5090-3491-8.

[30] Srikanth, Akash, a další. *Congestion Control Mechanism in Software Defined Networking by Traffic Rerouting.* Erode, India : IEEE, 2018. Second International Conference on Computing Methodologies and Communication (ICCMC). ISBN 978-1-5386-3452-3.

[31] Ferguson, Andrew D., a další. *Hierarchical policies for software defined networks.* Helsinki, Finland : ACM, 2012. Proceedings of the first workshop on Hot topics in software defined networks. ISBN 978-1-4503-1477-0.

[32] Ferguson, Andrew D., a další. *Participatory Networking: An API for Application Control of SDNs.* Helsinki, Finland : ACM, 2013. ACM SIGCOMM Conference. ISBN 978-1-4503-2056-6.

[33] Li, Yong a Chen, Min. *Software-Defined Network Function Virtualization: A Survey.* 2015. IEEE Access. vol. 3. ISSN 2169-3536.

[34] Li, Guanglei, Zhou, Huachun a Feng, Bohao. *Context-Aware Service Function Chaining and Its Cost-Effective Orchestration in Multi-Domain Networks.* Beijing, China: IEEE, 2018. Special Section on Survivability Strategies for Emerging Wireless Networks. ISSN 2169-3536

[35] Xing, Xiaodong, a další. *A defense mechanism against the DNS amplification attack in SDN.* Beijing, China: IEEE, 2016. IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC). ISBN 978-1-5090-1245-9.

[36] Nanda, Saurav, a další. *Predicting network attack patterns in SDN using machine learning approach.* Palo Alto, USA: IEEE, 2016. Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). ISBN 978-1-5090-0933-6.

[37] Yan, Qiao, a další. *Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges.* 2016. IEEE Communications Surveys & Tutorials. ISSN 1553-877X.

[38] Sahay, Rishikesh, a další. *Adaptive Policy-driven Attack Mitigation in SDN.* Belgrade, Serbia: ACM, 2017. Proceedings of the 1st International Workshop on Security and Dependability of Multi-Domain Infrastructures. ISBN 978-1-4503-4937-6.

[39] Kampanakis, Panos, Perros, Harry a Beyene, Tsegereda. *SDN-based solutions for Moving Target Defense network protection.* Sydney, Australia: IEEE, 2014. Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014. ISBN 978-1-4799-4786-7.

[40] Steinberger, Jessica, a další. *DDoS defense using MTD and SDN.* Taipei, Taiwan: IEEE, 2018. NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium. ISSN 2374-9709.

[41] Sama, Malla Reddy, a další. *Software-defined control of the virtualized mobile packet core.* 2015. IEEE Communications Magazine. vol. 53. ISSN 0163-6804.

[42] Filho, Roberto Irajá Tavares da Costa, Lautenschläger, William a Kagami, Nicolas. *Scalable QoE-aware Path Selection in SDN-based Mobile Networks.* Honolulu, USA: IEEE, 2018. INFOCOM 2018. ISBN 978-1-5386-4128-6.

[43] Gonsalves, Antone. Juniper unveils Penta chipset for 5G infrastructure. *TechTarget.* [Online] 15. 06. 2018 [cit. 2019-2-22]. Dostupné z: https://searchnetworking.techtarget.com/news/252443187/Juniper-unveils-Penta-chipset-for-5G-infrastructure.

[44] Switch Light. *Big Switch Networks.* [Online] [cit. 2019-2-28]. Dostupné z: https://www.bigswitch.com/solutions/technology/switch-light.

[45] Buck, Amy. OpenSwitch OPX Wiki. *GitHub.* [Online] [cit. 2019-3-4]. Dostupné z: https://github.com/open-switch/opx-docs/wiki.

[46] Tootoonchian, Amin, Gorbunov, Sergey a Ganjali, Yashar. *On Controller Performance in Software-Defined Networks.* San Jose, USA: ACM, 2012. Hot-ICE'12 Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. s. 10

[47] Erickson, David. *The Beacon OpenFlow Controller.* Stanford,CA: Stanford University, 2015. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking s. 13 -18. ISBN 978-1-4503-2178-5

[48] Bannour, Fetia, Souihi, Sami a Mellouk, Abdelhamid. *Distributed SDN Control: Survey, Taxonomy, and Challenges.* Paris,France: IEEE, 2018. IEEE Communications Surveys & Tutorials. vol. 20. ISSN 1553-877X.

[49] Ryu SDN Framework. Build SDN Agilely. *Ryu Github.* [Online] 2017. [cit. 2019-2-18]. Dostupné z: https://osrg.github.io/ryu/.

[50] Platform Overview. *OpenDaylight Foundation.* [Online] OpenDaylight Project a Series of LF Projects, 2018. [cit. 18-2-2019]. Dostupné z: https://www.opendaylight.org/what-we-do/odl-platform-overview.

[51] About the project. *ONOS Project.* [Online] [cit. 2019-2-18]. Dostupné z: https://onosproject.org/mission/.

[52] *HP VAN SDN Controller Programming Guide.* HP Development Company. 2014. ISSN 5998-6079.

[53] Tootoonchian, Amin a Ganjali, Yashar. *HyperFlow: A Distributed Control Plane for OpenFlow.* San Jose, USA: USENIX, 2010. Internet Network Management Workshop. s. 3.

[54] Shin, Seungwon, Song, Yongoo a Lee, Taekyung. *Rosemary: A Robust, Secure, and High-Performance Network Operating System.* Scottsdale,USA: ACM, 2014. ACM SIGSAC Conference on Computer and Communications Security. ISBN 978-1-4503-2957-6.

[55] Matsumoto, Stephanos, Hitz, Samuel a Perrig, Adrian. *Fleet: Defending SDNs from Malicious Administrators.* Chicago, USA: ACM, 2014. HotSDN'14. ISBN 978-1-4503-2989-7.

[56] Phemius, Kévin, Bouet, Mathieu a Leguay, Jérémie. *DISCO: Distributed multi-domain SDN controllers.* Krakow, Poland: IEEE, 2014. 2014 IEEE Network Operations and Management Symposium (NOMS). ISBN 978-1-4799-0913-1.

[57] Trois, Celio, a další. *A Survey on SDN Programming Languages: Toward a Taxonomy.* Vitoria, Brazil: IEEE, 2016. IEEE Communications Surveys & Tutorials. ISSN 1553-877X.

[58] Monsanto, Christopher, a další. *Composing Software-Defined Networks.* Lombard, USA : USENIX, 2013. USENIX conference on Networked Systems Design and Implementation. Dostupné z: https://www.usenix.org/system/files/conference/nsdi13/nsdi13-final232.pdf

[59] Soulé, Robert, a další. *Merlin: A Language for Managing Network Resources.* Sydney, Australia: ACM, 2014. 10th ACM International on Conference on emerging Networking Experiments and Technologies. ISBN 978-1-4503-3279-8.

[60] OpFlex: An Open Policy Protocol White Paper. *Cisco Solutions.* [Online] 2014. [cit. 2019-3-6]. Dostupné z: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html. 1538025281906783.

[61]The P4 Language Consortium. P4 Language Specification. [Online] 2018. [cit. 2019-3-6]. Dostupné z: https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.html.

[62] Li, Shengru, a další. *Protocol Oblivious Forwarding (POF): Software-Defined Networking with Enhanced Programmability.* Hefei, China: IEEE, 2017. IEEE Network. ISSN: 0890-8044.

[63] Big Switch Networks. *Open Network Linux.* [Online] 2018. [cit. 2019-2-28]. Dostupné z: https://www.bigswitch.com/solutions/technology/open-network-linux.

[64] Attention World: The Network Is Now Open! *Big Switch Networks.* [Online] [cit. 2019-2-28]. Dostupné z: https://www.bigswitch.com/blog/2015/03/09/attention-world-the-network-is-now-open.

[65] Rottenkolber, Max. Snabb: Simple and fast packet networking. *GitHub.* [Online] [cit. 2019-3-4]. Dostupné z: https://github.com/snabbco/snabb.

[66] Choi, Sean, a další. *FBOSS: Building Switch Software at Scale.* Budapest, Hungary: ACM, 2018. SIGCOMM '18. ISBN 978-1-4503-5567-4.

[67] Stratum – ONF Launches Major New Open Source SDN Switching Platform with Support from Google. *Open Networking Foundation.* [Online] 12. 3. 2018 [cit. 2019-3-7]. Dostupné z: https://www.opennetworking.org/news-and-events/press-releases/stratum-onf_google-launches-major-new-open-source-sdn-switching-platform-with-support-from-google/.

[68] Finucane, Stephen. Why Open vSwitch? *OVS Github.* [Online] 12. 12. 2016 [cit. 2019-4-18]. Dostupné z: https://github.com/openvswitch/ovs/blob/master/Documentation/intro/why-ovs.rst.

[69] Zanna, Paul. Zodiac FX. *NorthBound Networks.* [Online] [cit. 2019-4-18]. Dostupné z: https://northboundnetworks.com/collections/zodiac-fx/products/zodiac-fx.

[70] Ljubicic, Marko. Ryu-SDN-Load-Balancer. *Github.* [Online] 24. 10. 2017 [cit. 2019-4-19]. Dostupné z: https://github.com/exploitthesystem/Ryu-SDN-Load-Balancer.

[71] Lin, John. Pigrelay: Open Unix Domain Socket for Snort(NIDS). *GitHub.* [Online] 6. 7. 2015. [cit. 2019-4-20]. Dostupné z: https://github.com/John-Lin/pigrelay.

[72] L2_firewall. *Kickstart SDN.* [Online] 01. 04. 2015. [cit. 2019-4-30] Dostupné z: https://kickstartsdn.com/l2_firewall/.

Přílohy

Laboratorní úloha 1 – Vyvažování zátěže

Cíle laboratorní úlohy

- Seznámit se s možnostmi sítí SDN
- Provést základní konfiguraci síťových prvků a kontroléru
- Procvičit si základní příkazy systému *Linux* a základy jazyka *Python*
- Úspěšně nakonfigurovat kontrolér pro vyvažování zátěže a ověřit jeho funkčnost

Potřebné pomůcky

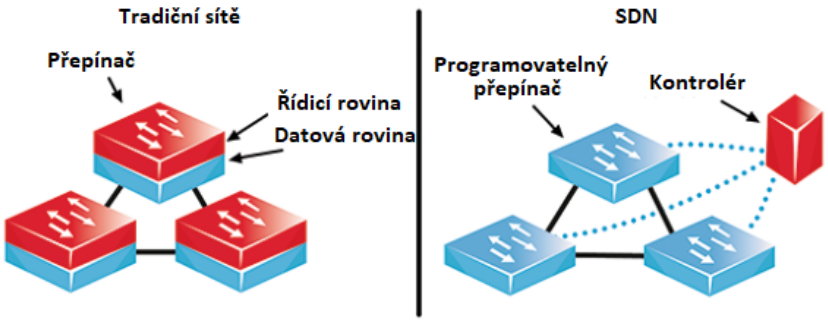
- 2x *RaspberryPi 3B+* s předinstalovaným OS *Raspbian* a spuštěným SSH serverem
- 2x osobní počítač s programem *Virtualbox, SSH klientem (*například *Putty)* a dvěma nezávislými ethernetovými rozhraními
- 4x USB síťová karta
- router sloužící jako brána pro připojení k Internetu a správu pomocí SSH
- připravený obraz OS pro *Virtualbox* (například *ArchLinux*)
- program pro vyvažování zátěže

Předpokládané znalosti

- Základní práce se systémem *Linux*
- Funkce jednotlivých komponent (kontrolér, *Open vSwitch*, vyvažování zátěže)
- Konfigurace virtuálních klientů v programu *Virtualbox*

Problematika

SDN (*Software defined network*) je nový přístup k architektuře, který slibuje větší dynamiku, flexibilitu a schopnost adaptovat se na současný síťový provoz. Klasický síťový prvek se obecně skládá ze 2 rovin, z řídicí a datové. Řídicí rovina se stará o sledování a řízení toků paketů, které skutečně zpracovává nižší, datová rovina. V SDN dochází k oddělení datové roviny síťového prvku od řídicí roviny, která tak může být přístupná k úpravám a přímo ovlivnitelná přes tzv. API (*Application Programming Interface*). Nabízí tak možnost síťovému administrátorovi zasahovat do řízení toků paketů. Díky centralizaci řídící roviny a globálnímu přehledu o stavu sítě lze efektivněji řídit datový provoz a přizpůsobovat síťové parametry na základě různorodých požadavků. Mezi ty nejčastější patří zpoždění, fázové chvění, chybovost, přenosová rychlost, úroveň zabezpečení, dostupnost aj.



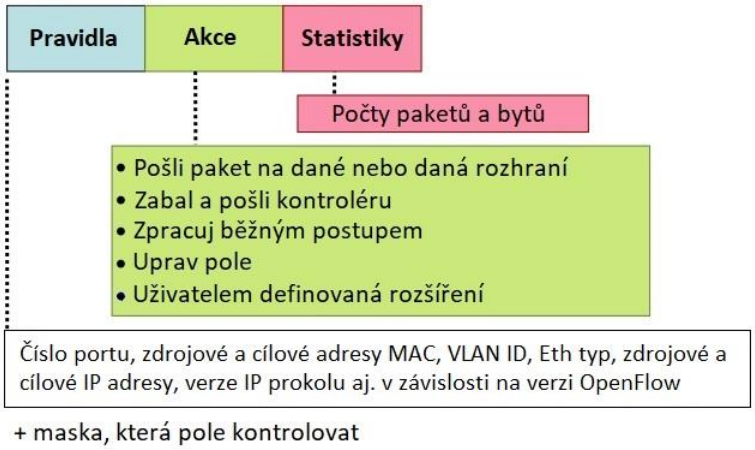
Obrázek 1.1 - Rozdíl mezi klasickou architekturou a SDN

Openflow je jedním z nejrozšířenějších otevřených komunikačních standardů mezi řídicí a datovou vrstvou v SDN. Jeho základ je postaven na technologii *ethernet* s využitím „*flow*" tabulek a definovaných akcí.

Kontrolér spolu s přepínači komunikují pomocí zabezpečeného kanálu, přičemž komunikaci zahajuje přepínač. Pro bezpečný přenos informací využívají TLS spojení na portu 6653. K autentifikaci dochází za pomoci výměny certifikátů a jejich ověření pomocí privátních klíčů. *Openflow* označuje porty na příchozí straně jako vstupní (*inbound)* a straně odchozí jako (*outbound*).

Po přijetí paketu na vstupním portu, dochází k rozhodování na základě porovnání se záznamy v první „*flow*" tabulce. Pokud nedojde ke shodě, přepínač porovnává hlavičku paketu s dalšími tabulkami, která má každá své unikátní ID. Tabulek může být až 255. V případě nenalezení shody (*table-miss*) dochází k vyřazení „*flow*", přesměrování na kontrolér, popřípadě k zaplavování více rozhraní. Pokud je nalezena shoda dochází k provedení přiřazených akcí, inkrementaci statistických údajů a směrování „*flow*" na dané výstupní rozhraní.

Tabulky *Openflow* se skládají z těchto hlavních částí: pravidla, akce a statistiky. Pravidla se skládají z podmíněných vlastností jako jsou například příchozí port, MAC a IP adresy, čísla TCP rozhraní, příslušnosti k dané VLAN aj. V závislosti na verzi OF se přidávají i položky priorita, další instrukce, „*timeout* „či „*cookies".*



Obrázek 1.2 - Struktura *OpenFlow* tabulky

Open vSwitch je virtuální přepínač poskytující široké portfolio funkcionalit, rozhraní pro správu a nativní podporu *Openflow*. Díky využití virtualizace a *Open vSwitche* můžeme v úloze využít *RaspberryPi* jako plnohodnotný SDN přepínač.

Skládá se z několika důležitých komponent:

- *Ovs-vswitchd* je program běžící na pozadí, který realizuje samotný přepínač.
- *Ovs-dpctl* je nástroj pro konfiguraci jádra virtualizovaného přepínače.
- *Ovs-vsctl* obsahuje nástroje pro přístup a úpravy konfigurace přepínače.
- Ovs-ofctl je nástrojem pro komunikaci a kontrolu *OpenFlow* přepínačů.

RYU, vydaný pod licencí *Apache* 2.0, je kontrolér napsaný v *Pythonu* a jeho velkou výhodou je plná podpora OF ve verzích 1.0 až 1.5. Jeho základní filozofií je schopnost fungovat jako modulární rámec obsahující různé stavební kameny, namísto rozsáhlého kontroléru. Pro naši úlohu bude důležitá nativní aplikace *simple_switch_13.py.*

Vyvažování zátěže (*load balancing*) je obecně technika pro rozkládání zátěže mezi dva a více spoje, počítače, či procesory. V našem případě se jedná o vytěžování serverů, které obsluhují požadavky připojených uživatelů. Díky rozložení zátěže dokáží servery v závislosti na spojení a dalších nejen přenosových parametrech obsloužit v ideálním případě až jednou tolik požadavků. Mezi nejzákladnější techniky patří schéma nazývané „*round-robin",* v němž dochází k postupnému střídání obsluhujících serverů popořadě a pořád dokola.

Doporučená literatura pro další studium

Open vSwitch Documentation: First Steps [online]. Linux Foundation Collaborative Project, 2016 [cit. 2019-04-20]. Dostupné z: https://docs.Open vSwitch.org/en/latest/

What's Ryu: Getting Started [online]. Nippon Telegraph and Telephone Corporation, 2014 [cit. 2019-04-20]. Dostupné z: https://ryu.readthedocs.io/en/latest/getting_started.html

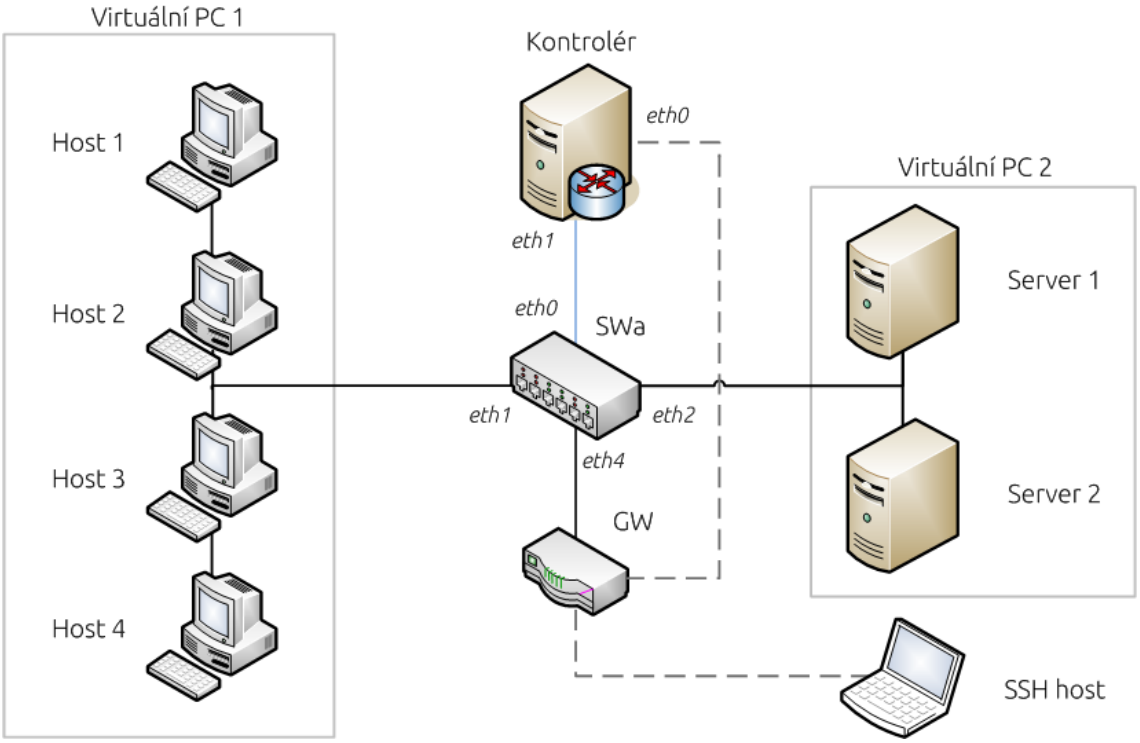
Ryu-SDN-Load-Balancer. Marko Ljubicic. *Github.* [online] 24. 10 2017. Dostupné z: https://github.com/exploitthesystem/Ryu-SDN-Load-Balancer

Úkol laboratorní úlohy

Demonstrujte si na dostupných zařízeních možnost realizace SDN sítě. Nakonfigurujte úlohu podle zadání a ověřte, zda dochází k úspěšnému vyvažování zátěže mezi servery.

Scénář

Je dána jednoduchá topologie vyobrazená na obrázku 1.3. Využijte kontrolér *Ryu* pro efektivnější využívání výpočetního výkonu obou serverů. Nakonfigurujte síť tak, aby bylo docíleno vyvažování zátěže mezi dvěma připojenými servery. Při konfiguraci využijte adresy uvedené v tabulce 1.1. Účastníky připojující se na server i samotné servery virtualizujte pomocí programu *Virtualbox* a připravených obrazů OS *Archlinux*. Jako poskytovanou službu využijte například jednoduchý http server.



Obrázek 1.3 - Schéma zapojení laboratorní úlohy č. 1

Tabulka 1.1 - Navrhované adresy pro úlohu 1 převzaté z kapitoly o stavbě platformy

Kontrolér		**SWa**	
Port	**Adresa**	**Port**	**Adresa**
eth0	*192.168.0.10*	eth0	*192.168.1.11*
eth1	*192.168.1.10*	eth1	*Síťový most*
eth2	*192.168.2.10*	eth2	
eth3	*192.168.3.10*	eth3	
eth4	*192.168.4.10*	eth4	*192.168.0.11*
Podsíť LAN (správa a přístup k Internetu)		*192.168.0.0*	
Brána		*192.168.0.1*	
SSH host		*192.168.0.250*	
Podsíť pro připojená zařízení		*192.168.100.0*	

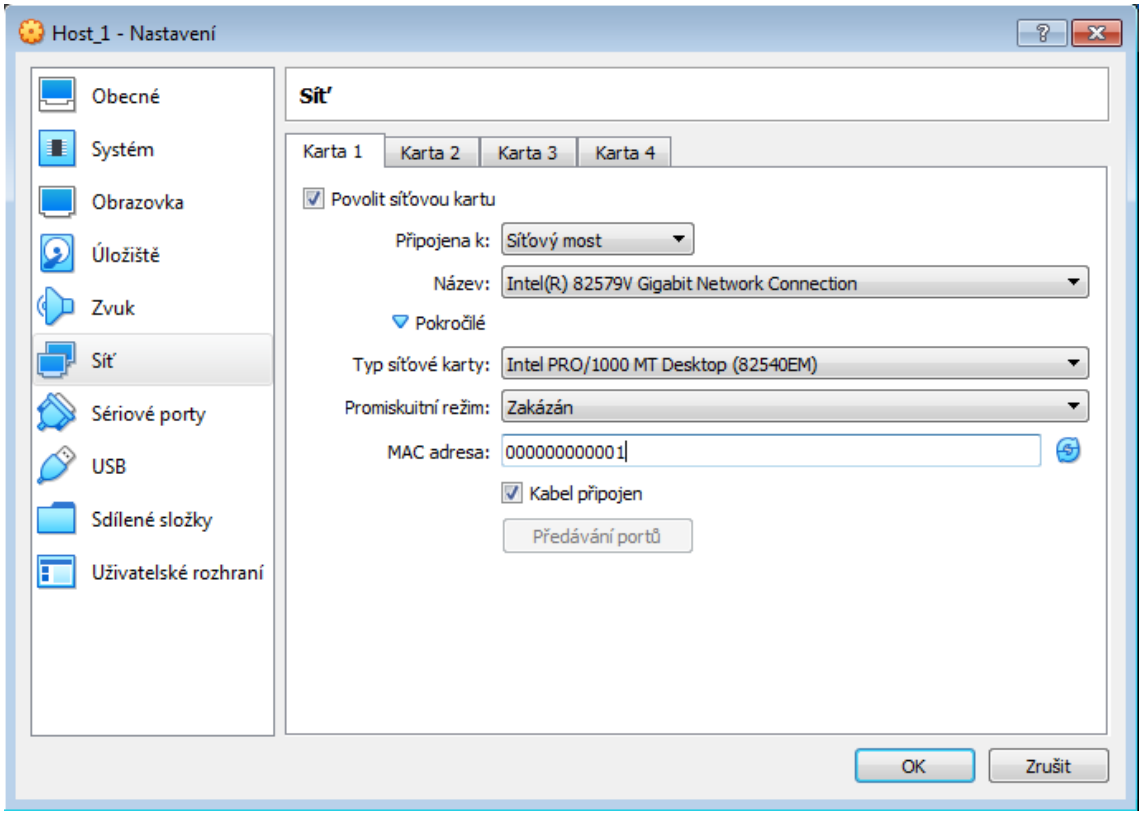
Doporučený postup

Prvním krokem je zapojení USB síťových karet a celého pracoviště podle schématu na obrázku 1.3.

Virtuální stanice

Pro vytvoření nového virtuálního stroje je nutné v levém horním rohu programu *Virtualbox* kliknout na tlačítko *nový*. V následujícím dialogu uvedeme *název* (například *Host_1*), vybereme *typ OS (Linux)* a *verzi (ArchLinux 64-bit).* V dalším kroku přidělíme *operační paměť* (postačí *1024 MB*), vytvoříme *virtuální úložiště* (*například uvedených 8 GB*). Dále ponecháme výchozí hodnoty *typu souboru* a *dynamicky alokovaný prostor*. Po ukončení úvodního dialogu nového zařízení se nám stroj objeví ve sloupci na levé straně.

Dále je nutné nastavit síťovou kartu virtuálního stroje. To provedeme otevřením *nastavení* právě přidaného OS. V záložce *síť* vybereme *první adaptér* a změníme jeho *typ* na *síťový most*, což umožní virtualizovaným systémům vystupovat přes fyzické rozhraní počítače pod svojí adresou. V *pokročilých volbách* dále změníme *adresu MAC* na dobře čitelnou *00:00:00:00:00:0X*.
Pro urychlené přidání dalších klientů je vhodné využít možnost klonovat, dostupnou po kliknutím pravým tlačítkem myši na daný virtuální stroj.



Obrázek 1.4 - Vzorové nastavení sítě v programu *Virtualbox* pro prvního klienta

Po spuštění virtuálních strojů je nutné zadat cestu k obrazu daného OS. Po naběhnutí systému nastavíme IP adresy. Ty byly pro uživatele zvoleny v rozsahu *192.168.100.101 – 192.168.100.104* a pro servery *192.168.100.105* a *192.168.100.106*.

Pro nastavení adres využijeme příkaz:

```
sudo ifconfig NÁZEV_ROZHRANÍ 192.168.100.10x
```

Konfigurace kontroléru

Ve výchozím nastavení *Raspbian* získává IP adresu z DHCP serveru, který je v našem případě provozován bránou *GW*. Pro přístup k zařízení použijeme SSH protokol. Výchozími přihlašovacími údaji jsou uživatel *pi* s heslem *raspberry*.

Nejsnazší cestou k získání kontroléru je stažení přes manažer *Python* balíčků PIP.

```
sudo apt-get install python-dev python3-pip
sudo pip3 install ryu
```

Dále nakonfigurujeme síťová rozhraní kontroléru v `etc/network/interfaces` podle tabulky 1.1. Pro úpravu použijeme například editor *nano*.

```
auto eth0
iface eth0 inet static
    address 192.168.0.10
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
    dns-server 8.8.8.8

auto eth1
iface eth1 inet static
    address 192.168.1.10
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
```

Dále provedeme vypnutí DHCP služby příkazem:

```
sudo update-rc.d -f dhcpcd remove
```

V případě potřeby jej lze opětovně zapnout příkazem:

```
sudo update-rc.d -f dhcpcd defaults
```

Po restartu zařízení bude dále dostupné pro správu pomocí SSH pod adresou *192.168.0.10*.

Program pro kontrolér [70] je možné snadno získat z *„githubu“* pomocí příkazu:

```
git clone https://github.com/exploitthesystem/Ryu-SDN-Load-
Balancer/blob/master.
```

Následně je výhodné jej nakopírovat do složky s aplikacemi pro Ryu: `/usr/local/lib/python2.7/dist-packages/ryu/app`.

Konfiguraci provedeme změnou parametrů ve staženém souboru *LoadBalancer.py.* Obrázek 1.5 ukazuje proměnné obsahující jednotlivé adresy. Při prozkoumání zdrojového kódu aplikace *LoadBalancer* vidíme, že přijímá požadavky od uživatelů na levé straně a podle schématu *round-robin* střídá servery 1 a 2, přičemž nahrazuje cílové a zdrojové adresy dotazů tak, aby spolu obě strany mohly komunikovat napřímo.

```
class LoadBalancer(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]     # Specify the use of OpenFlow v13
    virtual_ip = "192.168.100.100"              # The virtual server IP
    ## Hosts 5 and 6 are servers.
    H5_mac = "00:00:00:00:00:05"          # Host 5's mac
    H5_ip = "192.168.100.105"                   # Host 5's IP
    H6_mac = "00:00:00:00:00:06"          # Host 6's mac
    H6_ip = "192.168.100.106"                   # Host 6's IP
    next_server = ""       # Stores the IP of the  next server to use in round robin manner
    current_server = ""    # Stores the current server's IP
    ip_to_port = {H5_ip: 5, H6_ip: 6}
    ip_to_mac = {"192.168.100.101": "00:00:00:00:00:01",
                 "192.168.100.102": "00:00:00:00:00:02",
                 "192.168.100.103": "00:00:00:00:00:03",
                 "192.168.100.104": "00:00:00:00:00:04"}
```

Obrázek 1.5 - Konfigurace programu pro vyvažování zátěže

Konfigurace přepínače

Nejprve je nutné stáhnout *Open vSwitch*. Například jako již zkompilovaný balíček pomocí příkazu:

```
sudo apt-get install bridge-utils OpenvSwitch-switch OpenvSwitch-
common
```

Dalším krokem je konfigurace rozhraní na přepínačích. Následuje vzorová konfigurace v `etc/network/interfaces` pro *SWa*, která rozhraním přiřazuje adresy podle tabulky 1.1.

```
#SWa
allow-hotplug eth0
iface eth0 inet static
    address 192.168.1.11
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
    gateway 192.168.1.10
    dns-server 8.8.8.8

allow-hotplug eth1
iface eth1 inet manual

allow-hotplug eth2
iface eth2 inet manual

allow-hotplug eth3
iface eth3 inet manual
```

```
allow-hotplug eth4
iface eth4 inet static
    address 192.168.0.11
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
    dns-server 8.8.8.8

auto BRa
allow-ovs BRa
iface BRa inet manual
    ovs_type OVSBridge
    ovs_ports eth1 eth2 eth3
```

Poslední odstavec popisuje *Open vSwitch* síťový most *BRa*, který bude dostupný i po restartu zařízení a bude obsahovat rozhraní *eth1-3*.

Samotný síťový most ale musí být nadefinován pomocí následujících příkazů. Program *ovs-vsctl* poskytuje uživateli rozhraní pro přístup ke konfigurační databázi přepínače. Po přiřazení požadovaných rozhraní je nutné dále přidat adresu kontroléru a příkazem `connection-mode=out-of-band` říci přepínači, že kontrolér bude komunikovat po vyhrazeném rozhraní, namísto vnitřních rozhraní přepínače.

```
sudo ovs-vsctl add-br BRa
sudo ovs-vsctl set bridge BRa protocols=Openflow10, Openflow13
sudo ifconfig BRa up
sudo ovs-vsctl add-port BRa eth1
sudo ovs-vsctl add-port BRa eth2
sudo ovs-vsctl add-port BRa eth3
sudo ovs-vsctl set-controller BRa tcp:192.168.1.10:6633
sudo ovs-vsctl set controller BRa connection-mode=out-of-band
```

Po přidání síťového mostu, je záhodné jeho správné přidání zkontrolovat příkazem `ifconfig` nebo lépe `sudo ovs-ofctl show`, který vypisuje současnou konfiguraci OF přepínače.

```
pi@SWa:~ $ sudo ovs-vsctl show
a2558bf3-3a1b-4d0a-8202-777b8605437f
    Bridge BRa
        Controller "tcp:192.168.1.10:6633"
        Port BRa
            Interface BRa
                type: internal
        Port "eth2"
            Interface "eth2"
        Port "eth3"
            Interface "eth3"
        Port "eth1"
            Interface "eth1"
    ovs_version: "2.3.0"
pi@SWa:~ $
```

Obrázek 1.6 - Očekávaný výpis správné konfigurace *Open vSwitch* síťového mostu *BRa*

Spuštění úlohy

Před samotným spuštěním kontroléru spustíme na serveru 1 a 2 jednoduchý http server pomocí příkazu: `python -m http.server 80`, který od *Pythonu* verze 3 nahrazuje často používaný příkaz `python -m SimpleHTTPServer`.

Na straně kontroléru spustíme 2 aplikace, a to *simple_switch_13* zajišťující přepínání a *LoadBalancer*, pomocí příkazu:
`ryu-manager --verbose ryu.app.simple_switch_13 ryu.app.LoadBalancer`

Na straně uživatelů se například pomocí příkazu `wget -O 192.168.100.100` snažíme o stažení obsahu webového serveru.

Při opakování předchozího příkazu na uživatelských zařízeních 1-4 bychom kromě úspěšných spojení a vypsání obsahu serveru měli vidět záznam o příchozích příkazech *GET* na serveru 1 a 2.

V okně kontroléru by se při správné funkci měly vypisovat záznamy o událostech z aplikace vyvažování zátěže, na kterých je vidět směrování dotazů na servery a pravidelné střídání jejich adres.

Požadovaný výstup

Demonstrujte vedoucímu cvičení správnou funkci úlohy. Vhodným způsobem je předložení výpisů konzolí z uživatelských terminálů 1 až 4 a zprávy o úspěšných spojeních z konzole serverů 1 a 2. Dále na událostech získaných z *„logu“* kontroléru popište funkci vyvažování zátěže.

Pokuste se zodpovědět následující otázky:

- Má použitý způsob nějaká omezení?
- Jaký je efektivnější způsob řešení, než pomocí schématu *round-robin?*
- Jak je vyvažování zátěže řešeno v klasických sítích?

Závěr

Úloha umožňuje úspěšně nakonfigurovat kontrolér pro jednoduchý příklad vyvažování zátěže mezi dvěma servery. Nabízí tak první setkání s SDN a použitou laboratorní platformou.

Laboratorní úloha 1 – Verze pro vyučujícího

Úloha je primárně určena pro samostatnou činnost jednotlivců. V případě nedostatečného počtu pomůcek je možné pracovat v menších skupinkách.

Celková odhadovaná doba přípravy: 50 min
Celková odhadovaná doba potřebná pro vypracování: 90 min

Cíle laboratorní úlohy

- seznámit se s možnostmi sítí SDN
- provést základní konfiguraci síťových prvků a kontroléru
- procvičit si základní příkazy systému *Linux* a základy jazyka *Python*
- úspěšně nakonfigurovat kontrolér pro vyvažování zátěže a ověřit jeho funkčnost

Potřebné vybavení

- 2x *RaspberryPi 3B+* s předinstalovaným OS *Raspbian* a spuštěným SSH serverem
- 2x osobní počítač s programem *Virtualbox, SSH klientem (například Putty)* a dvěma nezávislými ethernetovými rozhraními
- 4x USB síťová karta
- směrovač sloužící jako brána pro připojení k Internetu a správu pomocí SSH
- připravený obraz OS pro *Virtualbox* (například *ArchLinux*)
- program pro vyvažování zátěže

Příprava

- paměťová karta s nahraným OS *Raspbian* (*15 min*)
- instalace programů *Virtualbox* a *Putty* (*10 min*)
- distribuce obrazu *Archlinuxu* na všechny stanice, popřípadě na FTP server (*5 min*)
- nastavení směrovače pro umožnění správy a přístupu do Internetu (*15 min*)
- kontrola konektivity a dostupnosti potřebných souborů (5 min)

Postup

1. vytvoření virtuálních strojů (*15 min*)
2. zapojení úlohy podle schématu (*5 min*)
3. nastavení adres (*15 min*)
4. instalace kontroléru a nastavení programu (*10 min*)
5. konfigurace přepínače (*15 min*)
6. spuštění vyvažování (*15 min*)
7. ústní kontrola dokončení úloh a zodpovězení otázek (*5 min*)

Nejčastější problémy

Špatné zapojení (přehozená čísla rozhraní)

- zkontrolovat správné zapojení

Základní konektivita (špatně nakonfigurované IP adresy)

- překontrolovat příkazem: `ifconfig`
- zkusit *„ping“* mezi jednotlivými zařízeními

Neschopnost navázání spojení kontroléru s přepínačem

- zkontrolovat konfiguraci síťového mostu příkazem: `sudo ovs-ofctl show`
- prověřit výpis *„logu“ Open vSwitch* pomocí příkazu:
  `tail -f /var/log/Open vSwitch/ovs-vswitchd.log`

Studenti neznají základní příkazy pro práci s OS Linux

- Zopakovat relevantní příkazy jako `cd, cp, ifconfig` aj.
- Zopakovat systém oprávnění a příkaz `sudo`

Doporučená literatura pro řešení dalších problémů

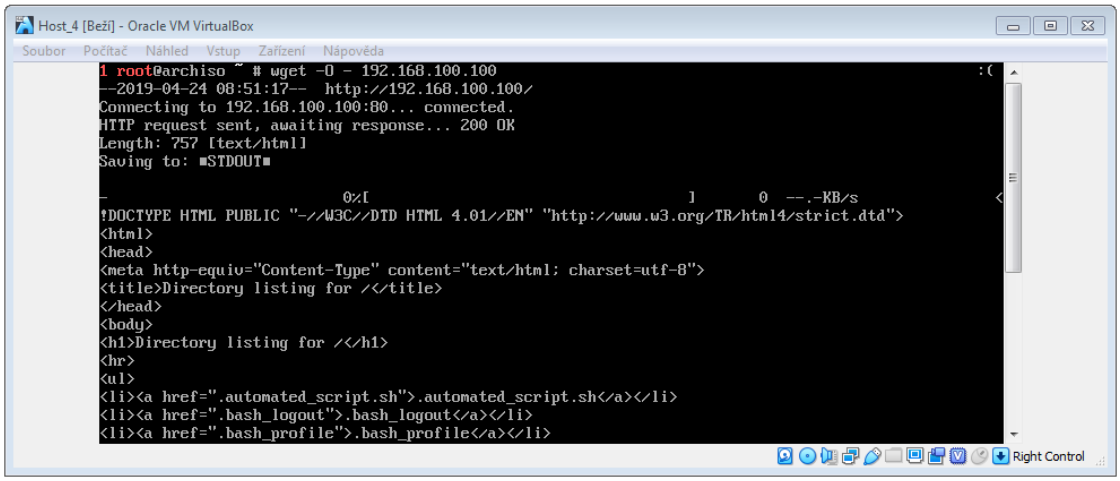
Open vSwitch Documentation: First Steps [online]. Linux Foundation Collaborative Project, 2016 [cit. 2019-04-20]. Dostupné z: https://docs.Open vSwitch.org/en/latest/

What's Ryu: Getting Started [online]. Nippon Telegraph and Telephone Corporation, 2014 [cit. 2019-04-20]. Dostupné z: https://ryu.readthedocs.io/en/latest/getting_started.html

Ryu-SDN-Load-Balancer. Marko Ljubicic. *Github.* [online] 24. 10 2017. Dostupné z: https://github.com/exploitthesystem/Ryu-SDN-Load-Balancer

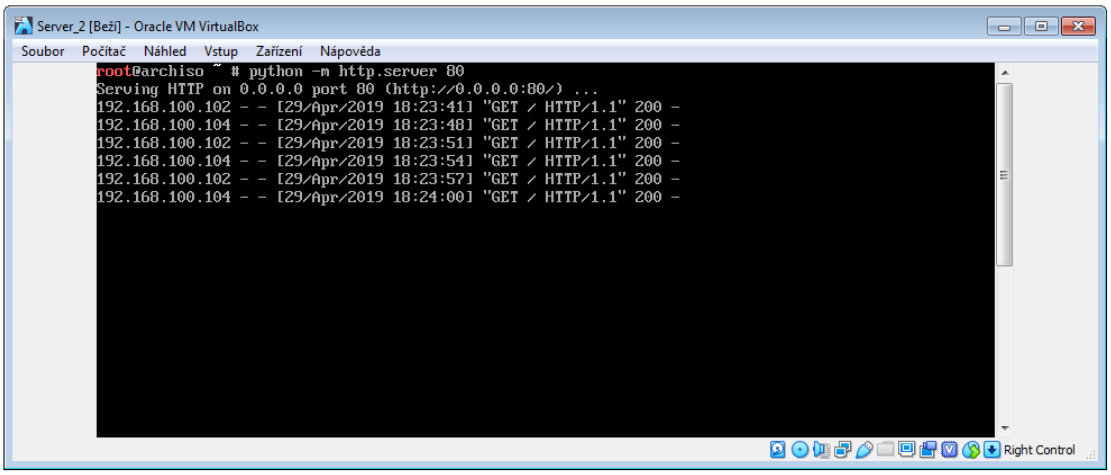
Požadovaný výstup

Funkce celé úlohy dokládají následující obrázky. Obrázek 1.1 ukazuje navázání spojení účastníka 4 se serverem, potvrzení zprávou 200 OK a zobrazení samotného obsahu načteného dokumentu.



Obrázek 1.1 - Dotaz na virtuální server

Obrázek 1.2 zobrazuje výpis ze serveru 2 o úspěšných příkazech *GET* od účastníků číslo 2 a 4. Dále pak obrázek 1.3 ukazuje přepínání mezi servery 1 a 2 při zasílání požadavků od hostů v pořadí od 1 do 4.



Obrázek 1.2 - Výpis serveru 2 o přijetí *GET* příkazů

```
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:01 00:00:00:00:00:05 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:01 00:00:00:00:00:05 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:02 00:00:00:00:00:06 2
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:02 00:00:00:00:00:06 2
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:03 ff:ff:ff:ff:ff:ff 3
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:03 00:00:00:00:00:05 3
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:01 00:00:00:00:00:05 1
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:03 00:00:00:00:00:05 3
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:04 ff:ff:ff:ff:ff:ff 4
EVENT ofp_event->SimpleSwitch13 EventOFPPacketIn
EVENT ofp_event->LoadBalancer EventOFPPacketIn
packet in 1 00:00:00:00:00:04 00:00:00:00:00:06 4
```

Obrázek 1.3 - Výpis konzole aplikace vyvažování zátěže na kontroléru

Laboratorní úloha 2 – Analýza provozu pomocí IDS *Snort*

Cíle laboratorní úlohy

- Seznámit se s možnostmi sítí SDN
- Ověřit možnost nasazení IDS *Snort* na *RaspberryPi*
- Provést základní konfiguraci síťových prvků a kontroléru
- Procvičit si základní příkazy systému *Linux* a základy jazyka *Python*
- Úspěšně nakonfigurovat kontrolér pro spolupráci s IDS *Snort*

Potřebné pomůcky

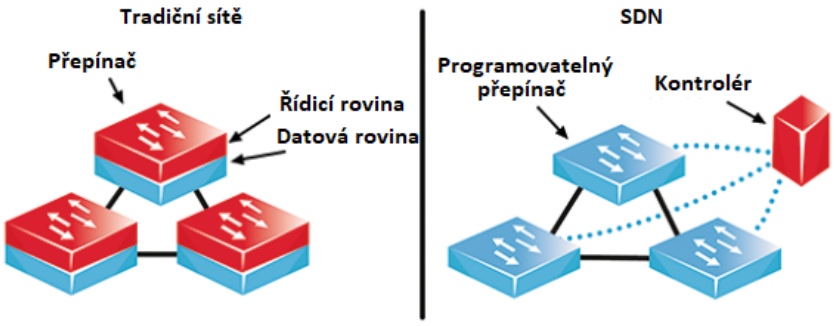
- 3x *RaspberryPi 3B+* s předinstalovaným OS *Raspbian* a spuštěným SSH serverem
- 2x osobní počítač s SSH klientem *(*například *Putty)*
- 5x USB síťová karta
- router sloužící jako brána pro připojení k Internetu a správu pomocí SSH

Předpokládané znalosti

- Základní práce se systémem *Linux*
- Funkce jednotlivých komponent (kontrolér, *Open vSwitch*, IDS *Snort*)

Problematika

SDN (*Software defined network*) je nový přístup k architektuře, který slibuje větší dynamiku, flexibilitu a schopnost adaptovat se na současný síťový provoz. Klasický síťový prvek se obecně skládá ze 2 rovin, z řídicí a datové. Řídicí rovina se stará o sledování a řízení toků paketů, které skutečně zpracovává nižší, datová rovina. V SDN dochází k oddělení datové roviny síťového prvku od řídicí roviny, která tak může být přístupná k úpravám a přímo ovlivnitelná přes tzv. API (*Application Programming Interface*). Nabízí tak možnost síťovému administrátorovi zasahovat do řízení toků paketů. Díky centralizaci řídicí roviny a globálnímu přehledu o stavu sítě lze efektivněji řídit datový provoz a přizpůsobovat síťové parametry na základě různorodých požadavků. Mezi ty nejčastější patří zpoždění, fázové chvění, chybovost, přenosová rychlost, úroveň zabezpečení, dostupnost aj.



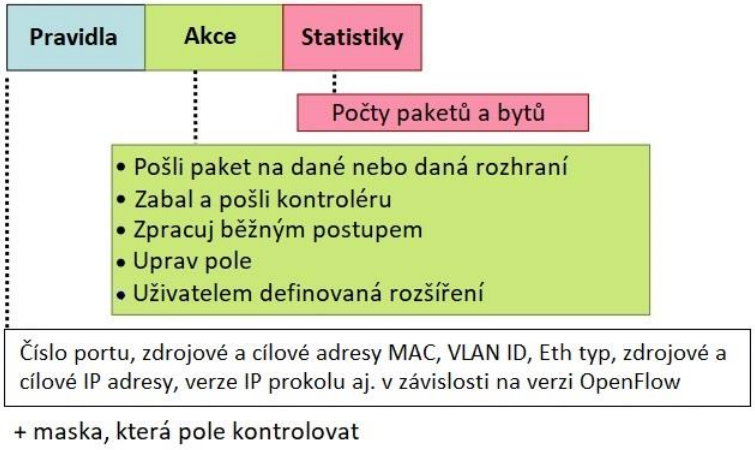
Obrázek 2.1 - Rozdíl mezi klasickou architekturou a SDN

Openflow je jedním z nejrozšířenějších otevřených komunikačních standardů mezi řídicí a datovou vrstvou v SDN. Jeho základ je postaven na technologii *Ethernet* s využitím „*flow*“ tabulek a definovaných akcí.

Kontrolér spolu s přepínači komunikují pomocí zabezpečeného kanálu, přičemž komunikaci zahajuje přepínač. Pro bezpečný přenos informací využívají TLS spojení na portu 6653. K autentifikaci dochází za pomoci výměny certifikátů a jejich ověření pomocí privátních klíčů. *Openflow* označuje porty na příchozí straně jako vstupní (*inbound)* a straně odchozí jako (*outbound*).

Po přijetí paketu na vstupním portu, dochází k rozhodování na základě porovnání se záznamy v první „*flow*“ tabulce. Pokud nedojde ke shodě, přepínač porovnává hlavičku paketu s dalšími tabulkami, která má každá své unikátní ID. Tabulek může být až 255. V případě nenalezení shody (*table-miss*) dochází k vyřazení „*flow*“, přesměrování na kontrolér, popřípadě k zaplavování více rozhraní. Pokud je nalezena shoda dochází k provedení přiřazených akcí, inkrementaci statistických údajů a směrování „*flow*“ na dané výstupní rozhraní.

Tabulky *Openflow* se skládají z těchto hlavních částí: pravidla, akce a statistiky. Pravidla se skládají z podmíněných vlastností jako jsou například příchozí port, MAC a IP adresy, čísla TCP rozhraní, příslušnosti k dané VLAN aj. V závislosti na verzi OF se přidávají i položky priorita, další instrukce, „*timeout“* či „*cookies“.*



Obrázek 2.2 - Struktura *OpenFlow* tabulky

Open vSwitch je virtuální přepínač poskytující široké portfolio funkcionalit, rozhraní pro správu a nativní podporu *Openflow*. Díky využití virtualizace a *Open vSwitche* můžeme v úloze využít *RaspberryPi* jako plnohodnotný SDN přepínač.

Skládá se z několika důležitých komponent:

- *Ovs-vswitchd* je program běžící na pozadí, který realizuje samotný přepínač.
- *Ovs-dpctl* je nástroj pro konfiguraci jádra virtualizovaného přepínače.
- *Ovs-vsctl* obsahuje nástroje pro přístup a úpravy konfigurace přepínače.
- Ovs-ofctl je nástrojem pro komunikaci a kontrolu *OpenFlow* přepínačů.

***RYU*,** vydaný pod licencí *Apache* 2.0, je kontrolér napsaný v *Pythonu* a jeho velkou výhodou je plná podpora OF ve verzích 1.0 až 1.5. Jeho základní filozofií je schopnost fungovat jako modulární rámec obsahující různé stavební kameny, namísto rozsáhlého kontroléru. Pro naši úlohu bude důležitá nativní aplikace *simple_switch_snort.py.*

Zrcadlení portů *(port mirroring)* je schopnost zařízení, v našem případě virtuálního přepínače, veškerý provoz na určitém rozhraní kopírovat na rozhraní jiné.

IDS (*Intrusion detection system*) se snaží na základě monitorování probíhajícího provozu odhalit podezřelé, či nechtěné aktivity. V závislosti na definovaných pravidlech a nastavených prazích se jedná například o sbírání informací před útokem, skenování portů, snahu o zjištění topologie, či samotné útoky.

Snort patří mezi pasivní síťové IDS zaměřené na charakteristické projevy chování. Pasivním chováním se vyznačují IDS, které aktivně nezasahují do síťového provozu a v případě podezřelého chování pouze generují upozornění. Skupina aktivních, označovaných jako IPS (*Intrusion prevention system*), na daný útok reagují například zablokováním dané služby či spojení.

Charakteristické chování je definováno pomocí pravidel, která určují parametry, po jejichž překročení je odesláno upozornění. Pro správnou funkci je důležitá správná definice a odladění prahových hodnot pro konkrétní síť. Samotný *Snort* v sobě obsahuje bohatou základnu pravidel pro nejrůznější typy útoků.

Doporučená literatura pro další studium

Open vSwitch Documentation: First Steps [online]. Linux Foundation Collaborative Project, 2016 [cit. 2019-04-20]. Dostupné z: https://docs.Open vSwitch.org/en/latest/

What's Ryu: Getting Started [online]. Nippon Telegraph and Telephone Corporation, 2014 [cit. 2019-04-20]. Dostupné z: https://ryu.readthedocs.io/en/latest/getting_started.html

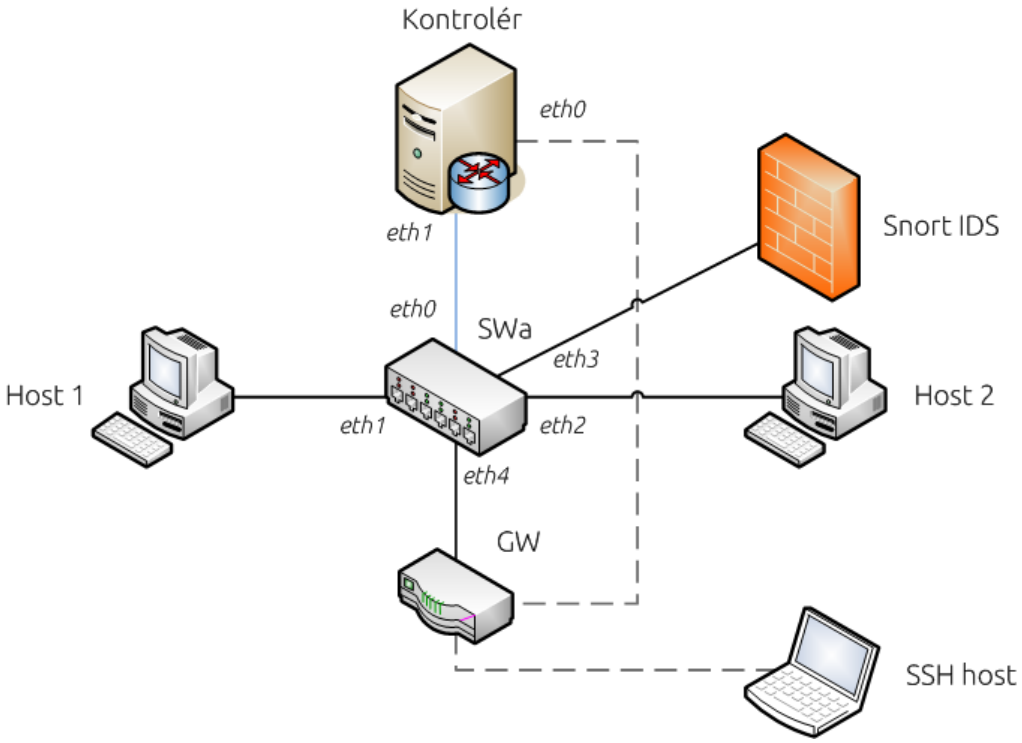
SNORT Users Manual: 2.9.13 [online]. The Snort Project, 2019 [cit. 2019-04-20]. Dostupné z: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/

Úkol laboratorní úlohy

Demonstruje si na připravených zařízeních schopnost kontroléru spolupracovat s IDS *Snort*. Nakonfigurujte úlohu podle zadání a ověřte pomocí testovacího pravidla, zdali dochází k předávání upozornění na kontrolér.

Scénář

Na dané topologii z obrázku 2.3 zprovozněte bezpečnostní analýzu provozu pomocí IDS *Snort*. Pro jeho implementaci využijte jedno z přiložených *RaspberryPi.* Nakonfigurujte IDS tak, aby zprávy o závadném provozu odesílalo na kontrolér, který je vypíše do své konzole. Pro ověření funkce využijte jednoduché pravidlo pro zaznamenání provozu ICMP protokolu. ICMP provoz vytvořte například prostým příkazem *„ping“* z jedné uživatelské stanice na druhou.



Obrázek 2.3 - Schéma zapojení laboratorní úlohy č. 2

Tabulka 2.1 - Navrhované adresy pro úlohu 2 převzaté z kapitoly o stavbě platformy

Kontrolér		**SWa**	
Port	**Adresa**	**Port**	**Adresa**
eth0	*192.168.0.10*	eth0	*192.168.1.11*
eth1	*192.168.1.10*	eth1	*Síťový most*
eth2	*192.168.2.10*	eth2	
eth3	*192.168.3.10*	eth3	
eth4	*192.168.4.10*	eth4	*192.168.0.11*
Podsíť LAN (správa a přístup k Internetu)		*192.168.0.0*	
Brána		*192.168.0.1*	
SSH host		*192.168.0.250*	
Podsíť pro připojená zařízení		*192.168.100.0*	

Doporučený postup

Prvním krokem je zapojení USB síťových karet a celého pracoviště podle schématu na obrázku 2.3.

Uživatelé

Nejprve provedeme nastavení IP adres uživatelských zařízení. V závislosti na použitém OS využijeme dostupnou grafickou nadstavbu nebo příkazový řádek. Jako konkrétní adresy zvolíme *192.168.100.100* pro uživatele jedna a *192.168.100.101* pro uživatele dvě.

V případě OS *Linux* pro nastavení adres využijeme příkaz:

```
sudo ifconfig NÁZEV_ROZHRANÍ 192.168.100.10x
```

Konfigurace kontroléru

Ve výchozím nastavení *Raspbian* získává IP adresu z DHCP serveru, který v našem případě provozuje brána *GW*. Pro přístup k zařízení použijeme SSH protokol. Výchozí přihlašovací údaje jsou uživatel *pi* s heslem *raspberry*.

Nejsnazší cestou k získání kontroléru je stažení přes manažer *Python* balíčků PIP.

```
sudo apt-get install python-dev python3-pip
sudo pip3 install ryu
```

Dále nakonfigurujeme síťová rozhraní kontroléru v `etc/network/interfaces` podle tabulky 2.1. Pro úpravu použijeme například editor *nano*.

```
auto eth0
iface eth0 inet static
    address 192.168.0.10
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
    dns-server 8.8.8.8

auto eth1
iface eth1 inet static
    address 192.168.1.10
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
```

Dále provedeme vypnutí DHCP služby příkazem:

```
sudo update-rc.d -f dhcpcd remove
```

V případě potřeby jej lze opětovně zapnout příkazem:

```
sudo update-rc.d -f dhcpcd defaults
```

Po restartu zařízení bude dále dostupné pro správu pomocí SSH pod adresou *192.168.0.10*.

Aplikaci *simple_switch_snort* je před spuštěním na kontroléru nutné nakonfigurovat tak, aby definovala správný port a naslouchala zprávám od *pigrelay.py*

To provedeme upravením souboru *simple_switch_snort.py* v umístění:
`/home/pi/.local/lib/python2.7/site-packages/ryu/app/`

Zde je nutné zvolit správné číslo portu, na kterém bude provozována služba IDS a povolit naslouchání varovných zpráv od IDS pomocí změny parametru „*unixsock*“ na „*false*“.

```
class SimpleSwitchSnort(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {'snortlib': snortlib.SnortLib}

    def __init__(self, *args, **kwargs):
        super(SimpleSwitchSnort, self).__init__(*args, **kwargs)
        self.snort = kwargs['snortlib']
        self.snort_port = 3
        self.mac_to_port = {}

        socket_config = {'unixsock': False}

        self.snort.set_config(socket_config)
        self.snort.start_socket_server()
```

Obrázek 2.4 - Konfigurace aplikace *Simple_switch_snort.py*

Konfigurace IDS Snort

Snort nainstalujeme na zbývající *RaspberryPi* z testovacího pracoviště příkazem:

```
sudo apt-get install Snort
```

Po instalaci vybereme v zobrazeném konfiguračním dialogu manuální metodu spouštění a adresní rozsah sítě pro kontrolu *192.168.0.0/16.*

Pro snadné ověření funkce přidáme do složky `/etc/snort/rules/` následující testovací pravidlo, které reaguje na jakoukoliv ICMP zprávu ve všech směrech komunikace.

```
pi@Pipi_Longstrocking: ~
 GNU nano 2.7.4                                  Soubor: /etc/snort/rules/Custom.rules

alert icmp any any -> any any (msg:"Testovaci_ping_alert";sid:1000004;)
```

Obrázek 2.5 - Testovací pravidlo pro *Snort*

Dále je nutné do konfiguračního souboru `/etc/snort/snort.conf` přidat cestu k nově vytvořenému souboru s pravidlem příkazem `include $RULE_PATH/Custom.rules`.

Pro vypisování zachycených varování na kontroléru je nutné na zařízení se *Snort* spustit na pozadí službu, která tyto zprávy poslouchá na *Unixovém „socketu“* a dále je po síti přeposílá na *Ryu*. Zdrojový kód je dostupný na *„githubu“* [71].

Ten se stáhne příkazem: `git clone https://github.com/John-Lin/pigrelay`.
Dále jej spustíme v dalším okně příkazem: `sudo python pigrelay.py`.

Konfigurace přepínače

Nejprve je nutné stáhnout *Open vSwitch*. Například jako již zkompilovaný balíček pomocí příkazu:

```
sudo apt-get install bridge-utils OpenvSwitch-switch OpenvSwitch-
common
```

Dalším krokem je konfigurace rozhraní na přepínačích. Následuje vzorová konfigurace v `etc/network/interfaces` pro *SWa*, která rozhraním přiřazuje adresy podle tabulky 2.1.

```
#SWa
allow-hotplug eth0
iface eth0 inet static
    address 192.168.1.11
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
    gateway 192.168.1.10
    dns-server 8.8.8.8

allow-hotplug eth1
iface eth1 inet manual
allow-hotplug eth2
iface eth2 inet manual
allow-hotplug eth3
iface eth3 inet manual

allow-hotplug eth4
iface eth4 inet static
    address 192.168.0.11
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
    dns-server 8.8.8.8

auto BRa
allow-ovs BRa
iface BRa inet manual
    ovs_type OVSBridge
    ovs_ports eth1 eth2 eth3
```

Poslední odstavec popisuje *Open vSwitch* síťový most *BRa*, který bude dostupný i po restartu zařízení a bude obsahovat rozhraní *eth1-3*.

Samotný síťový most ale musí být nadefinován pomocí následujících příkazů. Program *ovs-vsctl* poskytuje uživateli rozhraní pro přístup ke konfigurační databázi přepínače. Po přiřazení požadovaných rozhraní je nutné dále přidat adresu kontroléru a příkazem `connection-mode=out-of-band` říci přepínači, že kontrolér bude komunikovat po vyhrazeném rozhraní, namísto vnitřních rozhraní přepínače.

```
sudo ovs-vsctl add-br BRa
sudo ovs-vsctl set bridge BRa protocols=Openflow10, Openflow13
sudo ifconfig BRa up
sudo ovs-vsctl add-port BRa eth1 -- set interface eth1
ofport_request=1
sudo ovs-vsctl add-port BRa eth2 -- set interface eth2
ofport_request=2
sudo ovs-vsctl add-port BRa eth3 -- --id=@port get port eth3 -- --
id=@mir create mirror name=m0 select-all=true output-port=@port -- set
bridge BRa mirrors=@mir -- set interface eth3 ofport_request=3

sudo ovs-vsctl set-controller BRa tcp:192.168.1.10:6633
sudo ovs-vsctl set controller BRa connection-mode=out-of-band
```

K tomu, aby byl OVS přepínač schopen zrcadlit provoz *BRa* na rozhraní 3, ke kterému je připojen *Snort*, je nutné rozšířit definici rozhraní přiřazených pod *BRa*.

Pro to, aby přepínač znal konkrétní čísla fyzických rozhraní, je nutné je přiřadit příkazem:

```
 -- set interface eth1 ofport_request=1
```

Příkaz uvedený pro *eth3* vytvoří *mirror* s názvem *m0*, který patří pod *BRa* a jako výstupní rozhraní označí rozhraní *eth3*.

Po přidání síťového mostu, je záhodné jeho správné přidání zkontrolovat příkazem `ifconfig` nebo lépe `sudo ovs-ofctl show`, který vypisuje současnou konfiguraci OF přepínače.

```
pi@SWa:~ $ sudo ovs-vsctl show
a2558bf3-3a1b-4d0a-8202-777b8605437f
    Bridge BRa
        Controller "tcp:192.168.1.10:6633"
        Port BRa
            Interface BRa
                type: internal
        Port "eth2"
            Interface "eth2"
        Port "eth3"
            Interface "eth3"
        Port "eth1"
            Interface "eth1"
    ovs_version: "2.3.0"
pi@SWa:~ $
```

Obrázek 2.6 - Očekávaný výpis správné konfigurace *Open vSwitch* síťového mostu *BRa*

Pro kontrolu zrcadlení je vhodné provést záchyt na zařízení se *Snort* pomocí příkazu: `sudo tcpdump -i eth0 -n`.

A zahájit komunikaci z uživatele 1 na uživatele 2 například nástrojem *ping*.

```
pi@Pipi_Longstrocking:~ $ sudo tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:22:45.451347 ARP, Request who-has 192.168.100.101 tell 192.168.100.100, length 46
11:22:45.461506 ARP, Request who-has 192.168.100.101 tell 192.168.100.100, length 46
11:22:45.461713 ARP, Request who-has 192.168.100.101 tell 192.168.100.100, length 46
11:22:45.461889 ARP, Request who-has 192.168.100.101 tell 192.168.100.100, length 46
```

Obrázek 2.7 - Výpis ukazující zrcadlení provozu z rozhraní eth1 na eth3

Spuštění úlohy

Dále spustíme aplikaci příkazem:
`ryu-manager --verbose ryu.app.simple_switch_snort`

A samotný *Snort* s několika parametry:
`sudo snort -i eth0 -A unsock -l /log -c /snort.conf`

Parametr: `-l` definuje ukládání logů do složky `/log`,
`-c` definuje cestu k souboru s konfigurací,
`-A unsock` – zprávy odesílá na *UNIX „socket“*, kde si ho přečte program *pigrelay*

Pro ověření funkce použijeme nástroj *ping* mezi uživatelskými zařízeními.

Požadovaný výstup

Demonstrujte vedoucímu cvičení správnou funkci úlohy. Vhodným způsobem je předložení výpisu konzole z kontroléru, na kterém jsou jednoznačně vidět záznamy o porušení námi dříve přidaného pravidla.

Prohlédněte si další pravidla ve složce `/etc/snort/rules/`. Navrhněte další možné způsoby ověření funkčnosti úlohy.

Pokuste se zodpovědět následující otázky:

- Má analýza paketů běžící na *RaspberryPi* nějaká omezení?
- Jak by vypadalo pravidlo pro FTP spojení z uživatele 1 na uživatele 2?

Závěr

Úloha umožňuje ověřit možnost implementovat IDS *Snort* v síti SDN tak, aby spolupracovala s kontrolérem *Ryu*. Nabízí tak první setkání s SDN a použitou laboratorní platformou.

Laboratorní úloha 2 – Verze pro vyučujícího

Úloha je primárně určena pro samostatnou činnost jednotlivců, v případě nedostatečného počtu pomůcek je možné pracovat v menších skupinkách.

Celková odhadovaná doba přípravy: 40 min
Celková odhadovaná doba potřebná pro vypracování: 80 min

Cíle laboratorní úlohy

- Seznámit se s možnostmi sítí SDN
- Ověřit možnost nasazení IDS *Snort* na *RaspberryPi*
- Provést základní konfiguraci síťových prvků a kontroléru
- Procvičit si základní příkazy systému *Linux* a základy jazyka *Python*
- Úspěšně nakonfigurovat kontrolér pro spolupráci s IDS *Snort*

Potřebné vybavení

- 3x *RaspberryPi 3B+* s předinstalovaným OS *Raspbian* a spuštěným SSH serverem
- 2x osobní počítač s *SSH klientem (*například *Putty)*
- 5x USB síťová karta
- router sloužící jako brána pro připojení k Internetu a správu pomocí SSH

Příprava

- paměťová karta s nahraným OS *Raspbian* (*15 min*)
- instalace programu *Putty* na osobní počítač (*5 min*)
- nastavení směrovače pro umožnění správy a přístupu do Internetu (*15 min*)
- kontrola konektivity a dostupnosti potřebných souborů (5 min)

Postup

1. nastavení adres osobních počítačů (*5 min*)
2. zapojení úlohy (*5 min*)
3. nastavení adres ostatních zařízení (*15 min*)
4. nastavení IDS *Snort* (15 min)
5. instalace kontroléru a nastavení programu (*10 min*)
6. konfigurace přepínače (*15 min*)
7. spuštění programu a ověření funkčnosti (*5 min*)
8. ústní ověření dokončení úlohy a zodpovězení otázek (*10 min*)

Nejčastější problémy

Špatné zapojení (přehozená čísla rozhraní)

- zkontrolovat správné zapojení

Základní konektivita (špatně nakonfigurované IP adresy)

- překontrolovat příkazem: `ifconfig`
- zkusit *„ping“* mezi jednotlivými zařízeními

Neschopnost navázání spojení kontroléru s přepínačem

- zkontrolovat konfiguraci síťového mostu příkazem: `sudo ovs-ofctl show`
- prověřit výpis *logu Open vSwitch* pomocí příkazu:
  `tail -f /var/log/Open vSwitch/ovs-vswitchd.log`

Nefunkční zrcadlení rozhraní

- zkontrolovat postup konfigurace zrcadlení, popřípadě odebrat síťový most a nakonfigurovat znovu

Neschopnost IDS detekovat ping (špatně definované pravidlo)

- ověřit správné zadání pravidla a jeho přidání do seznamu

Studenti neznají základní příkazy pro práci s OS Linux

- Zopakovat relevantní příkazy jako `cd, cp, ifconfig` aj.
- Zopakovat systém oprávnění a příkaz `sudo`

Doporučená literatura pro řešení dalších problémů

Open vSwitch Documentation: First Steps [online]. Linux Foundation Collaborative Project, 2016 [cit. 2019-04-20]. Dostupné z: https://docs.Open vSwitch.org/en/latest/

What's Ryu: Getting Started [online]. Nippon Telegraph and Telephone Corporation, 2014 [cit. 2019-04-20]. Dostupné z: https://ryu.readthedocs.io/en/latest/getting_started.html

SNORT Users Manual: 2.9.13 [online]. The Snort Project, 2019 [cit. 2019-04-20]. Dostupné z: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/

Požadovaný výstup

Ve výpisu z kontroléru na obrázku 2.1 je po prvotní inicializaci a navázání komunikace s přepínačem možné vidět informaci o porušení testovacího pravidla, a to konkrétně použitím nástroje *ping* z uživatele 1 na uživatele 2.

```
alertmsg: Testovaci_ping_alert
icmp(code=0,csum=19725,data=echo(data=array('B', [97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 1
08, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 97, 98, 99, 100, 101, 102, 103, 104, 105]),i
d=1,seq=78),type=8)
ipv4(csum=42562,dst='192.168.100.101',flags=0,header_length=5,identification=724,offset=0,option=None,p
roto=1,src='192.168.100.100',tos=0,total_length=60,ttl=128,version=4)
ethernet(dst='08:00:27:4b:c4:f2',ethertype=2048,src='08:00:27:4b:c4:f1')
alertmsg: Testovaci_ping_alert
icmp(code=0,csum=19725,data=echo(data=array('B', [97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 1
08, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 97, 98, 99, 100, 101, 102, 103, 104, 105]),i
d=1,seq=78),type=8)
ipv4(csum=42562,dst='192.168.100.100',flags=0,header_length=5,identification=724,offset=0,option=None,p
roto=1,src='192.168.100.101',tos=0,total_length=60,ttl=128,version=4)
ethernet(dst='08:00:27:4b:c4:f1',ethertype=2048,src='08:00:27:4b:c4:f2')
```

Obrázek 2.1 - Zpráva o zachycené komunikaci pomocí zpráv ICMP

Jak by vypadalo pravidlo pro FTP spojení z uživatele 1 na uživatele 2?

```
alert tcp 192.168.100.100 21 -> 192.168.100.101 any (msg: "Pokus o FTP
spojeni"; sid:xxxxxxxxx;)
```

Laboratorní úloha 3 – Řízení přístupu

Cíle laboratorní úlohy

- Seznámit se s možnostmi sítí SDN
- Seznámit se základní prací s kontrolérem POX
- Provést základní konfiguraci síťových prvků a kontroléru
- Úspěšně nakonfigurovat kontrolér pro řízení přístupu na základě MAC adres
- Procvičit si základní příkazy systému *Linux* a základy jazyka *Python*

Potřebné pomůcky

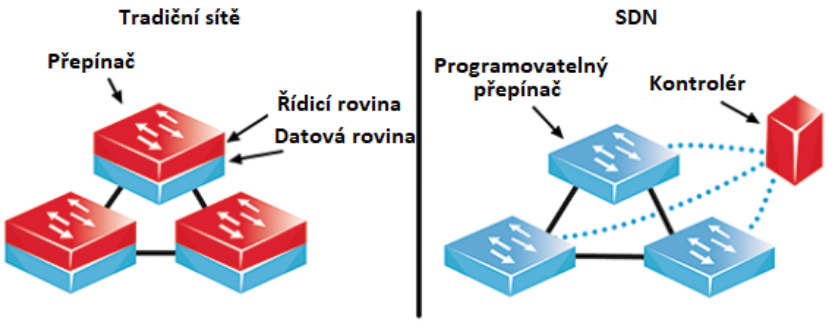
- 3x *RaspberryPi 3B+* s předinstalovaným OS *Raspbian* a spuštěným SSH serverem
- 2x osobní počítač s programem *Virtualbox, SSH klientem (například Putty)*
- 5x USB síťová karta
- směrovač sloužící jako brána pro připojení k Internetu a správu pomocí SSH
- připravený obraz OS pro *Virtualbox* (například *ArchLinux*)
- program pro řízení přístupu

Předpokládané znalosti

- Základní práce se systémem *Linux*
- Funkce jednotlivých komponent (kontrolér, *Open vSwitch*, *Virtualbox*)

Problematika

SDN (*Software defined network*) je nový přístup k architektuře, který slibuje větší dynamiku, flexibilitu a schopnost adaptovat se na současný síťový provoz. Klasický síťový prvek se obecně skládá ze 2 rovin, z řídicí a datové. Řídicí rovina se stará o sledování a řízení toků paketů, které skutečně zpracovává nižší, datová rovina. V SDN dochází k oddělení datové roviny síťového prvku od řídicí roviny, která tak může být přístupná k úpravám a přímo ovlivnitelná přes tzv. API (*Application Programming Interface*). Nabízí tak možnost síťovému administrátorovi zasahovat do řízení toků paketů. Díky centralizaci řídicí roviny a globálnímu přehledu o stavu sítě lze efektivněji řídit datový provoz a přizpůsobovat síťové parametry na základě různorodých požadavků. Mezi ty nejčastější patří zpoždění, fázové chvění, chybovost, přenosová rychlost, úroveň zabezpečení, dostupnost aj.



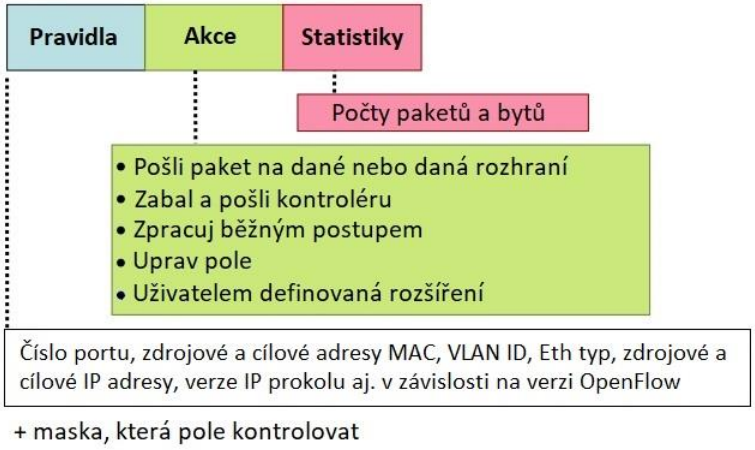
Obrázek 3.1 - Rozdíl mezi klasickou architekturou a SDN

Openflow je jedním z nejrozšířenějších otevřených komunikačních standardů mezi řídicí a datovou vrstvou v SDN. Jeho základ je postaven na technologii *ethernet* s využitím „*flow*“ tabulek a definovaných akcí.

Kontrolér spolu s přepínači komunikují pomocí zabezpečeného kanálu, přičemž komunikaci zahajuje přepínač. Pro bezpečný přenos informací využívají TLS spojení na portu 6653. K autentifikaci dochází za pomoci výměny certifikátů a jejich ověření pomocí privátních klíčů. *Openflow* označuje porty na příchozí straně jako vstupní (*inbound)* a straně odchozí jako (*outbound*).

Po přijetí paketu na vstupním portu, dochází k rozhodování na základě porovnání se záznamy v první „*flow*“ tabulce. Pokud nedojde ke shodě, přepínač porovnává hlavičku paketu s dalšími tabulkami, která má každá své unikátní ID. Tabulek může být až 255. V případě nenalezení shody (*table-miss*) dochází k vyřazení „*flow*“, přesměrování na kontrolér, popřípadě k zaplavování více rozhraní. Pokud je nalezena shoda dochází k provedení přiřazených akcí, inkrementaci statistických údajů a směrování „*flow*“ na dané výstupní rozhraní.

Tabulky *Openflow* se skládají z těchto hlavních částí: pravidla, akce a statistiky. Pravidla se skládají z podmíněných vlastností jako jsou například příchozí port, MAC a IP adresy, čísla TCP rozhraní, příslušnosti k dané VLAN aj. V závislosti na verzi OF se přidávají i položky priorita, další instrukce, „*timeout“* či „*cookies“.*



Obrázek 3.2 - Struktura *OpenFlow* tabulky

Open vSwitch je virtuální přepínač poskytující široké portfolio funkcionalit, rozhraní pro správu a nativní podporu *Openflow*. Díky využití virtualizace a *Open vSwitche* můžeme v úloze využít *RaspberryPi* jako plnohodnotný SDN přepínač.

Skládá se z několika důležitých komponent:

- *Ovs-vswitchd* je program běžící na pozadí, který realizuje samotný přepínač.
- *Ovs-dpctl* je nástroj pro konfiguraci jádra virtualizovaného přepínače.
- *Ovs-vsctl* obsahuje nástroje pro přístup a úpravy konfigurace přepínače.
- Ovs-ofctl je nástrojem pro komunikaci a kontrolu *OpenFlow* přepínačů.

POX je kontrolér s otevřeným zdrojovým kódem napsaný v jazyce *Python*. Jeho základy stojí na prvním kontroléru NOX**.** Nabízí více předpřipravených komponent pro výběr cest, správu topologie či rozšíření pro podporu *Open vSwitch* aj. Mimo OF podporuje i OVSDB protokol.

Řízení přístupu (*network access control*) je souhrn technik, protokolů a služeb využívaných pro zabezpečení přístupu koncových zařízení při prvním pokusu o připojení do sítě. Základním rozdělením je NAC s využitím agenta na koncové stanici, či bez něj a řízení přístupu před nebo po vpuštění do sítě. Dále se dělí podle toho, jak nakládají s porušením pravidel nebo zda využívají tzv. „*captive* portál". Mezi nejrozšířenější standardy pro řízení přístupu patří 802.1X.

Doporučená literatura pro další studium

Open vSwitch Documentation: First Steps [online]. Linux Foundation Collaborative Project, 2016 [cit. 2019-04-20]. Dostupné z: https://docs.Open vSwitch.org/en/latest/

MCCAULEY, Murphy. Github: POX Manual [online]. 2015 [cit. 2019-05-09]. Dostupné z: https://noxrepo.github.io/pox-doc/html/

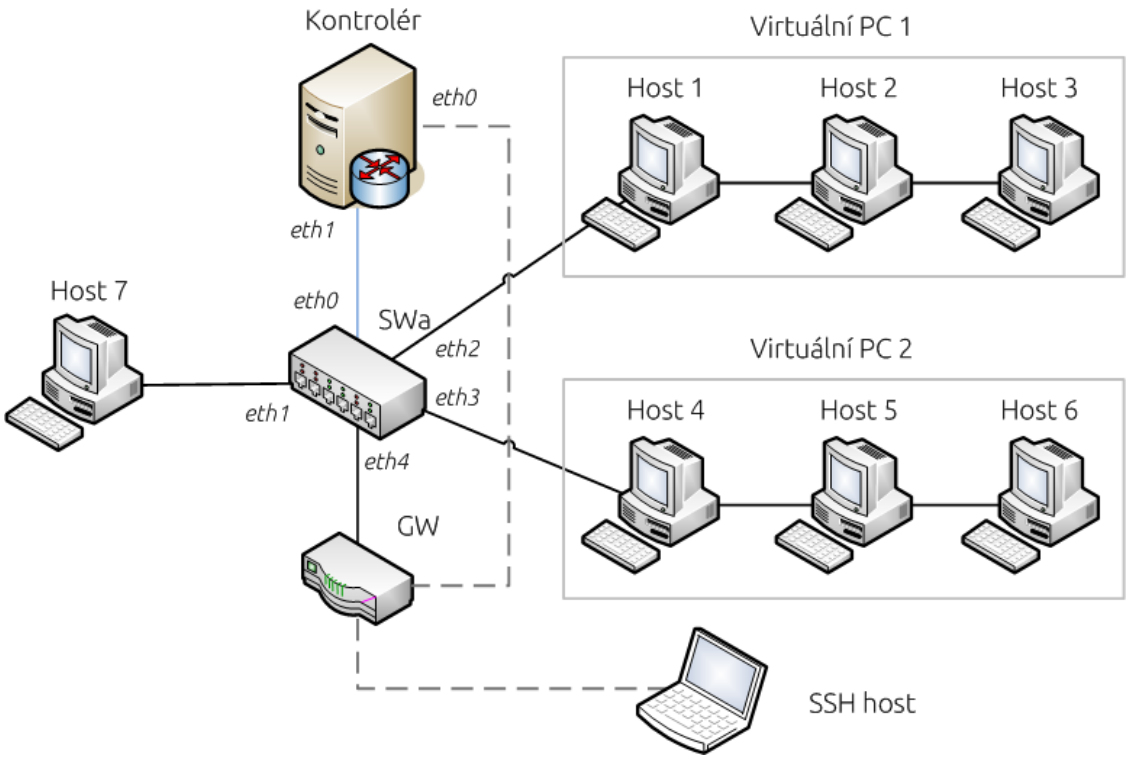
Oracle VM VirtualBox: User Manual [online]. Oracle Corporation, 2019 [cit. 2019-05-09]. Dostupné z: https://www.virtualbox.org/manual/

Úkol laboratorní úlohy

Demonstrujte na kontroléru POX možnost řídit přístup koncových zařízení na základě jejich MAC adresy. Pro realizaci úlohy využijte připravená zařízení. Ověřte neschopnost připojení zařízení mimo definovaný rozsah MAC adres.

Scénář

Na obrázku 3.3 je dána jednoduchá hvězdicová topologie reprezentující malou kancelářskou síť. Nakonfigurujte zařízení tak, aby byl zamezen přístup do sítě jiným než předdefinovaným zaměstnaneckým PC. Pro identifikaci zařízení použijte jednoduchý parametr, a to jejich MAC adresu. Ověřte funkci celé úlohy pomocí připojení dalšího zařízení a pokuste se o komunikaci se zaměstnaneckými PC například pomocí nástroje *ping*.



Obrázek 3.3 - Schéma zapojení laboratorní úlohy č. 3

Tabulka 3.1 - Navrhované adresy pro úlohu 3 převzaté z kapitoly o stavbě platformy

Kontrolér		**SWa**	
Port	**Adresa**	**Port**	**Adresa**
eth0	*192.168.0.10*	eth0	*192.168.1.11*
eth1	*192.168.1.10*	eth1	*Síťový most*
eth2	*192.168.2.10*	eth2	
eth3	*192.168.3.10*	eth3	
eth4	*192.168.4.10*	eth4	*192.168.0.11*
Podsíť LAN (správa a přístup k Internetu)		*192.168.0.0*	
Brána		*192.168.0.1*	
SSH host		*192.168.0.250*	
Podsíť pro připojená zařízení		*192.168.100.0*	

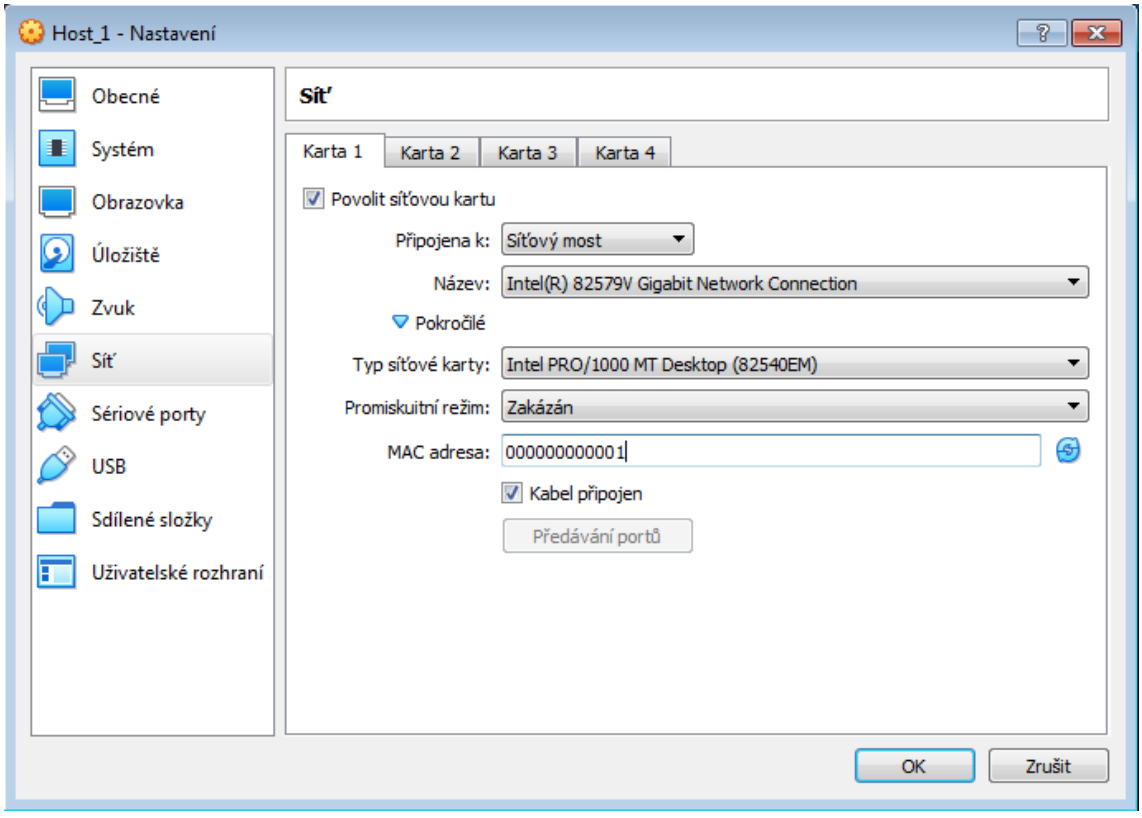
Doporučený postup

Prvním krokem je zapojení USB síťových karet a celého pracoviště podle schématu na obrázku 3.3.

Virtuální stanice

Pro vytvoření nového virtuálního stroje je nutné v levém horním rohu programu *Virtualbox* kliknout na tlačítko *nový*. V následujícím dialogu uvedeme *název* (například *Host_1*), vybereme *typ OS (Linux)* a *verzi (ArchLinux 64-bit).* V dalším kroku přidělíme *operační paměť* (postačí *1024 MB*), vytvoříme *virtuální úložiště* (*například uvedených 8 GB*), dále ponecháme *výchozí hodnoty typu souboru* a *dynamicky alokovaný prostor*. Po ukončení úvodního dialogu nového zařízení se nám stroj objeví ve sloupci na levé straně.

Dále je nutné nastavit síťovou kartu virtuálního stroje. To provedeme otevřením *nastavení* právě přidaného OS. V záložce *síť* vybereme *první adaptér* a změníme jeho *typ* na *síťový most*, což umožní virtualizovaným systémům vystupovat přes fyzické rozhraní počítače pod svojí adresou. V *pokročilých volbách* dále změníme *adresu MAC* na dobře čitelnou *00:00:00:00:00:0X*.
Pro urychlené přidání dalších klientů je vhodné využít možnost klonovat, dostupnou po kliknutím pravým tlačítkem myši na daný virtuální stroj.



Obrázek 3.4 - Vzorové nastavení sítě v programu *Virtualbox* pro prvního klienta

Po spuštění virtuálních strojů je nutné zadat cestu k obrazu daného OS. Po naběhnutí systému nastavíme IP adresy. Ty byly pro uživatele běžící na prvním PC v laboratoři zvoleny v rozsahu *192.168.100.101 – 192.168.100.103* a pro uživatele běžící na druhém PC *192.168.100.104 - 192.168.100.106*.

Pro nastavení adres využijeme příkaz:

```
sudo ifconfig NÁZEV_ROZHRANÍ 192.168.100.10x
```

Konfigurace kontroléru

Ve výchozím nastavení *Raspbian* získává IP adresu z DHCP serveru, který je v našem případě provozován bránou *GW*. Pro přístup k zařízení použijeme SSH protokol. Výchozími přihlašovacími údaji jsou uživatel *pi* s heslem *raspberry*.

Nejsnazší cestou k získání kontroléru je stažení přes manažer *Python* balíčků PIP.

```
sudo apt-get install python-dev python-pip
sudo pip install pox
```

Dále nakonfigurujeme síťová rozhraní kontroléru v `etc/network/interfaces` podle tabulky 3.1. Pro úpravu použijeme například editor *nano*.

```
auto eth0
iface eth0 inet static
    address 192.168.0.10
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
    dns-server 8.8.8.8

auto eth1
iface eth1 inet static
    address 192.168.1.10
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
```

Dále provedeme vypnutí DHCP služby příkazem:

```
sudo update-rc.d -f dhcpcd remove
```

V případě potřeby jej lze opětovně zapnout příkazem:

```
sudo update-rc.d -f dhcpcd defaults
```

Po restartu zařízení bude dále dostupné pro správu pomocí SSH pod adresou *192.168.0.10*.

Program pro kontrolér je možné získat z webu [72] pomocí příkazu:

```
wget  -O /usr/local/lib/python2.7/dist-packages/pox/ext/NAC.py
https://drive.google.com/file/d/0B7ulhq4nScO8bXhDUEZoOVJNRFk
```

Konfiguraci provedeme změnou parametrů ve staženém souboru *NAC.py.* Obrázek 3.5 ukazuje proměnné obsahující jednotlivé adresy. Při prozkoumání zdrojového kódu aplikace vidíme, že mimo třídy pro funkcionality samotného přepínače obsahuje také možnost definovat seznam konkrétních MAC adres. S těmi jsou později porovnávány MAC adresy, které se pokouší o komunikaci přes přepínač. V případě nenalezení shody je daný provoz zahazován. Voláním metody *AddRule postupně* přidáme MAC adresy zaměstnaneckých PC.

```
# Add a Couple of Rules
self.AddRule('00-00-00-00-00-01',EthAddr('00:00:00:00:00:01'))
self.AddRule('00-00-00-00-00-01',EthAddr('00:00:00:00:00:02'))
self.AddRule('00-00-00-00-00-01',EthAddr('00:00:00:00:00:03'))
self.AddRule('00-00-00-00-00-01',EthAddr('00:00:00:00:00:04'))
self.AddRule('00-00-00-00-00-01',EthAddr('00:00:00:00:00:05'))
self.AddRule('00-00-00-00-00-01',EthAddr('00:00:00:00:00:06'))
```

Obrázek 3.5 - Definice MAC adres pro udělení přístupu

Konfigurace přepínače

Nejprve je nutné stáhnout *Open vSwitch*. Například jako již zkompilovaný balíček pomocí příkazu:

```
sudo apt-get install bridge-utils OpenvSwitch-switch OpenvSwitch-
common
```

Dalším krokem je konfigurace rozhraní na přepínačích. Následuje vzorová konfigurace v `etc/network/interfaces` pro *SWa*, která rozhraním přiřazuje adresy podle tabulky 3.1.

```
#SWa
allow-hotplug eth0
iface eth0 inet static
    address 192.168.1.11
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
    gateway 192.168.1.10
    dns-server 8.8.8.8

allow-hotplug eth1
iface eth1 inet manual

allow-hotplug eth2
iface eth2 inet manual

allow-hotplug eth3
iface eth3 inet manual
```

```
allow-hotplug eth4
iface eth4 inet static
    address 192.168.0.11
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
    dns-server 8.8.8.8

auto BRa
allow-ovs BRa
iface BRa inet manual
    ovs_type OVSBridge
    ovs_ports eth1 eth2 eth3
```

Poslední odstavec popisuje *Open vSwitch* síťový most *BRa*, který bude dostupný i po restartu zařízení a bude obsahovat rozhraní *eth1-3*.

Samotný síťový most ale musí být nadefinován pomocí následujících příkazů. Program *ovs-vsctl* poskytuje uživateli rozhraní pro přístup ke konfigurační databázi přepínače. Po přiřazení požadovaných rozhraní je nutné dále přidat adresu kontroléru a příkazem `connection-mode=out-of-band` říci přepínači, že kontrolér bude komunikovat po vyhrazeném rozhraní, namísto vnitřních rozhraní přepínače.

```
sudo ovs-vsctl add-br BRa
sudo ovs-vsctl set bridge BRa protocols=Openflow10, Openflow13
sudo ifconfig BRa up
sudo ovs-vsctl add-port BRa eth1
sudo ovs-vsctl add-port BRa eth2
sudo ovs-vsctl add-port BRa eth3
sudo ovs-vsctl set-controller BRa tcp:192.168.1.10:6633
sudo ovs-vsctl set controller BRa connection-mode=out-of-band
```

Po přidání síťového mostu, je záhodné jeho správné přidání zkontrolovat příkazem `ifconfig` nebo lépe `sudo ovs-ofctl show`, který vypisuje současnou konfiguraci OF přepínače.

```
pi@SWa:~ $ sudo ovs-vsctl show
a2558bf3-3a1b-4d0a-8202-777b8605437f
    Bridge BRa
        Controller "tcp:192.168.1.10:6633"
        Port BRa
            Interface BRa
                type: internal
        Port "eth2"
            Interface "eth2"
        Port "eth3"
            Interface "eth3"
        Port "eth1"
            Interface "eth1"
    ovs_version: "2.3.0"
pi@SWa:~ $
```

Obrázek 3.6 - Očekávaný výpis správné konfigurace *Open vSwitch* síťového mostu *BRa*

Spuštění úlohy

Po přepnutí do složky se zdrojovým kódem kontroléru provedeme spuštění aplikace příkazem: `./pox.py log.level --DEBUG NAC`.

Při správné postupu a definování MAC adres dojde ke spuštění POX, načtení pravidel a kontrole probíhajících spojení.

```
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.16/Apr 6 2019 01:42:57)
DEBUG:core:Platform is Linux-5.0.0-13-generic-x86_64-with-Ubuntu-19.04-disco
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:l2_firewall:Connection [00-00-00-00-00-01 1]
DEBUG:l2_firewall:Adding firewall rule in 00-00-00-00-00-01: 00:00:00:00:00:01
DEBUG:l2_firewall:Adding firewall rule in 00-00-00-00-00-01: 00:00:00:00:00:02
DEBUG:l2_firewall:Adding firewall rule in 00-00-00-00-00-01: 00:00:00:00:00:03
DEBUG:l2_firewall:Adding firewall rule in 00-00-00-00-00-01: 00:00:00:00:00:04
DEBUG:l2_firewall:Adding firewall rule in 00-00-00-00-00-01: 00:00:00:00:00:05
DEBUG:l2_firewall:Adding firewall rule in 00-00-00-00-00-01: 00:00:00:00:00:06
```

Obrázek 3.7 - Načtení MAC adres kontrolérem

Otestujte správnou funkci řízení přístupu pomocí nástroje *„ping“* mezi virtuálními stroji na laboratorním PC 1 a 2. Při správné konfiguraci dochází k nalezení shody mezi MAC adresou zařízení a adresou v seznamu a úspěšnému přepínání provozu.

Pozn.: „*Ping*“ mezi virtuálními stroji v rámci jednoho PC nemá smysl, protože zde dochází k přepínání v rámci *Virtualboxu*.

Dále nastavte zbylému *RaspberryPi* IP adresu *192.168.100.107* a po připojení k *SWa* otestujte neschopnost spojení ke všem virtuálním strojům.

Požadovaný výstup

Demonstrujte vedoucímu cvičení správnou funkci úlohy. Vhodným způsobem je předložení výpisu konzole z kontroléru, na kterém jsou jednoznačně vidět záznamy o inicializaci kontroléru a úspěšném přepínání v rámci zaměstnaneckých PC. Dále doložte nemožnost spojení přidaného *RaspberryPi* reprezentující zařízení mimo majetek kanceláře.

Pokuste se zodpovědět následující otázky:

- Jak je možné tuto jednoduchou metodu autorizace obejít?
- Na základě jakých parametrů (či jejich kombinace) by bylo možné provádět řízení přístupu efektivněji?

Závěr

Úloha umožňuje ověřit schopnost kontroléru POX pracovat jako jednoduché NAC zařízení. Ověření uživatelů probíhá na základě sledování probíhajících spojení a filtrování použité MAC adresy zařízení.

Laboratorní úloha 3 – Verze pro vyučujícího

Úloha je primárně určena pro samostatnou činnost jednotlivců, v případě nedostatečného počtu pomůcek je možné pracovat v menších skupinkách.

Celková odhadovaná doba přípravy: 50 min
Celková odhadovaná doba potřebná pro vypracování: 80 min

Cíle laboratorní úlohy

- Seznámit se s možnostmi sítí SDN
- Seznámit se základní prací s kontrolérem POX
- Provést základní konfiguraci síťových prvků a kontroléru
- Úspěšně nakonfigurovat kontrolér pro řízení přístupu na základě MAC adres
- Procvičit si základní příkazy systému *Linux* a základy jazyka *Python*

Potřebné vybavení

- 3x *RaspberryPi 3B+* s předinstalovaným OS *Raspbian* a spuštěným SSH serverem
- 2x osobní počítač s programem *Virtualbox, SSH klientem (například Putty)*
- 5x USB síťová karta
- směrovač sloužící jako brána pro připojení k Internetu a správu pomocí SSH
- připravený obraz OS pro *Virtualbox* (například *ArchLinux*)
- program pro řízení přístupu

Příprava

- paměťová karta s nahraným OS *Raspbian* (*15 min*)
- instalace programů *Virtualbox* a *Putty* (*10 min*)
- distribuce obrazu *Archlinuxu* na všechny stanice, popřípadě na FTP server (*5 min*)
- nastavení směrovače pro umožnění správy a přístupu do Internetu (*15 min*)
- kontrola konektivity a dostupnosti potřebných souborů (5 min)

Postup

1. vytvoření virtuálních strojů (*15 min*)
2. zapojení úlohy podle schématu (*5 min*)
3. nastavení adres (*15 min*)
4. instalace kontroléru a nastavení programu (*10 min*)
5. konfigurace přepínače (*10 min*)
6. spuštění programu a ověření funkčnosti (*10 min*)
9. ústní kontrola dokončení úloh a zodpovězení otázek (*5 min*)

Nejčastější problémy

Špatné zapojení (přehozená čísla rozhraní)

- zkontrolovat správné zapojení

Základní konektivita (špatně nakonfigurované IP adresy)

- překontrolovat příkazem: `ifconfig`
- zkusit „*ping*“ mezi jednotlivými zařízeními

Neschopnost navázání spojení kontroléru s přepínačem

- zkontrolovat konfiguraci síťového mostu příkazem: `sudo ovs-ofctl show`
- prověřit výpis *logu Open vSwitch* pomocí příkazu:
  `tail -f /var/log/Open vSwitch/ovs-vswitchd.log`

Neschopnost kontroléru reagovat na provoz (špatně definované MAC adresy)

- *ověřit správnost v programu umístěném v*
  `/usr/local/lib/python2.7/dist-packages/pox/ext/NAC.py`

Studenti neznají základní příkazy pro práci s OS Linux

- Zopakovat relevantní příkazy jako `cd, cp, ifconfig` aj.
- Zopakovat systém oprávnění a příkaz `sudo`

Doporučená literatura pro řešení dalších problémů

Open vSwitch Documentation: First Steps [online]. Linux Foundation Collaborative Project, 2016 [cit. 2019-04-20]. Dostupné z: https://docs.Open vSwitch.org/en/latest/

MCCAULEY, Murphy. POX Manual: Github [online]. 2015 [cit. 2019-05-09]. Dostupné z: https://noxrepo.github.io/pox-doc/html/

Oracle VM VirtualBox: User Manual [online]. Oracle Corporation, 2019 [cit. 2019-05-09]. Dostupné z: https://www.virtualbox.org/manual/

Požadovaný výstup

Funkčnost úlohy dokládají následující obrázky. Obrázek 3.1 zobrazuje reakci kontroléru na povolený provoz. Obrázek 3.2 a obrázek 3.3 dokazují neschopnost vnějšího zařízení komunikovat s virtuálními stroji. Obrázek 3.4 ukazuje reakci kontroléru na neznámé zařízení.

```
DEBUG:l2_firewall:installing flow for 00:00:00:00:00:04.4 -> 00:00:00:00:00:01.1
DEBUG:l2_firewall:Rule (00:00:00:00:00:01) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:04.4
DEBUG:l2_firewall:Rule (00:00:00:00:00:04) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:installing flow for 00:00:00:00:00:04.4 -> 00:00:00:00:00:01.1
DEBUG:l2_firewall:Rule (00:00:00:00:00:02) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:Rule (00:00:00:00:00:05) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:installing flow for 00:00:00:00:00:05.5 -> 00:00:00:00:00:02.2
DEBUG:l2_firewall:Rule (00:00:00:00:00:02) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:05.5
DEBUG:l2_firewall:Rule (00:00:00:00:00:05) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:installing flow for 00:00:00:00:00:05.5 -> 00:00:00:00:00:02.2
DEBUG:l2_firewall:Rule (00:00:00:00:00:03) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:Rule (00:00:00:00:00:06) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:installing flow for 00:00:00:00:00:06.6 -> 00:00:00:00:00:03.3
DEBUG:l2_firewall:Rule (00:00:00:00:00:03) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:installing flow for 00:00:00:00:00:03.3 -> 00:00:00:00:00:06.6
DEBUG:l2_firewall:Rule (00:00:00:00:00:06) found in 00-00-00-00-00-01: FORWARD
DEBUG:l2_firewall:installing flow for 00:00:00:00:00:06.6 -> 00:00:00:00:00:03.3
```

Obrázek 3.1 - Záznam o přepínání *SWa* na základě nalezených MAC adres

```
pi@Pipi_Longstrocking:~ $ ping 192.168.100.101
PING 192.168.100.101 (192.168.100.101) 56(84) bytes of data.
From 192.168.100.10 icmp_seq=1 Destination Host Unreachable
From 192.168.100.10 icmp_seq=2 Destination Host Unreachable
From 192.168.100.10 icmp_seq=3 Destination Host Unreachable
^C
--- 192.168.100.101 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss, time 5191ms
```

Obrázek 3.2 - Neschopnost dosažení virtuálního stroje 1

```
pi@Pipi_Longstrocking:~ $ ping 192.168.100.105
PING 192.168.100.105 (192.168.100.105) 56(84) bytes of data.
From 192.168.100.10 icmp_seq=1 Destination Host Unreachable
From 192.168.100.10 icmp_seq=2 Destination Host Unreachable
From 192.168.100.10 icmp_seq=3 Destination Host Unreachable
^C
--- 192.168.100.105 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4143ms
```

Obrázek 3.3 - Neschopnost dosažení virtuálního stroje 2

```
DEBUG:l2_firewall:Rule (b8:27:eb:90:06:82) NOT found in 00-00-00-00-00-01: DROP
DEBUG:l2_firewall:Rule (b8:27:eb:90:06:82) NOT found in 00-00-00-00-00-01: DROP
DEBUG:l2_firewall:Rule (b8:27:eb:90:06:82) NOT found in 00-00-00-00-00-01: DROP
DEBUG:l2_firewall:Rule (b8:27:eb:90:06:82) NOT found in 00-00-00-00-00-01: DROP
DEBUG:l2_firewall:Rule (b8:27:eb:90:06:82) NOT found in 00-00-00-00-00-01: DROP
DEBUG:l2_firewall:Rule (b8:27:eb:90:06:82) NOT found in 00-00-00-00-00-01: DROP
```

Obrázek 3.4 - Zahazování provozu od neznámého zařízení