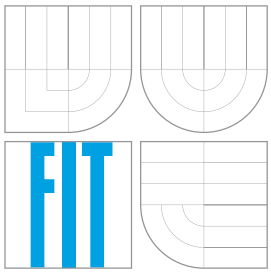
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SMĚROVÁNÍ SDN PODLE PŘENÁŠENÉHO OBSAHU

SDN ROUTING ACCORDING TO TRANSMITTED CONTENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

OLGA GAVRYLIUK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. HOLKOVIČ MARTIN

BRNO 2016

Zadání bakalářské práce/18916/2015/xgavry00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Gavryliuk Olga**
Obor: Informační technologie
Téma: **Směrování SDN podle přenášeného obsahu**
SDN Routing According to Transmitted Content
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se se softwarově definovanými sítěmi (SDN).
2. Navrhněte detekční mechanizmus pro systémy Windows, Linux a BSD, který na koncových zařízeních identifikuje procesy přistupující k síti.
3. Navrhněte aplikaci SDN řídicí směrování síťových toků využívající informace o procesech zohledňující jejich požadavky na šířku pásma a zpoždění.
4. Implementujte aplikace navržené v bodech dva a tři.
5. Otestujte vytvořené aplikace.
6. Zhodnoťte dosažené výsledky a navrhněte možná rozšíření.

Literatura:

- McKEOWN Nick, ANDERSON Tom, BALAKRISHNAN Hari, PARULKAR Guru, PETERSON Larry, REXFORD Jennifer, SHENKER Scott a TURNER Jonathan. OpenFlow: enabling innovation in campus networks. In: *SIGCOMM Comput. Commun. Rev.* 38, 2, 2008, s. 69-74.
- CALDAROLA Leo, CHOUKIR Amine, CUDA Davide, DONDERO Marco, FICARA Domenico, MUCCIFORA Roberto, POLČÁK Libor a TRIFILO Antonio. Towards a real application-aware network. In: Proceedings of the 6th International Conference on Data Communication Networking (DCNET-2015). Colmar: SciTePress - Science and Technology Publications, 2015, s. 5-12.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Holkovič Martin, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá směrováním v softwarově definovaných sítích, které umožňuje směrování podle obsahu přenášených paketů. Pro řešení práce bylo nutné vytvořit detekční mechanizmus, který v rámci operačního systému vyhledává spojení vyžadující prioritní zpracování. Nová spojení jsou odesílána směrovací aplikaci běžící na SDN kontroléru POX. Kontrolér rozděluje aplikace podle dvou kritérií: aplikace vyžadující největší šířku pásma a aplikace vyžadující nejrychlejší cestu. Byly navrhnuty a implementovány čtyři algoritmy pro směrování dat v závislosti na požadavcích aplikací. Navrhnuté aplikace byly implementovány v jazyce Python a otestovány emulačním nástrojem Mininet.

Abstract

This work's main point of focus is routing in software defined networks, which allow routing based on transfered packets. For accomplishing this task it was necessary to create a detection mechanism, that finds the connection within the OS which needs priority handling. New connections are sent via a routing aplication which is running on the SDN controller POX. The controller divides applications based on two criteria: the application requiring the largest bandwidth and the application requiring the fastest route. For this purpose four algorithms for routing data based on application preferences were designed and implemented. The applications were implemented in the Python language and they were tested in the simulation tool Mininet.

Klíčová slova

počítačové sítě, SDN, softwarově definované sítě, OpenFlow, QoE, směrování, SDN směrování, směrovací protokoly, POX

Keywords

computer networks, SDN, software-defined networks, OpenFlow, QoE, routing, SDN routing, routing protocols, POX

Citace

Olga Gavryliuk: Směrování SDN podle přenášeného obsahu, bakalářská práce, Brno, FIT VUT v Brně, 2016

Směrování SDN podle přenášeného obsahu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Martina Holkoviče

.....
Olga Gavryliuk
18. května 2016

Poděkování

Ráda bych poděkovala panu Ing. Martinovi Holkovičovi, za skvělé vedení této bakalářské práce, nezapomenutelnou vstřícnost, odbornou pomoc, vřelou podporu a cenné rady, které vedly k dosažení cíle.

Obsah

1 Úvod **2**

2 Softwarově definované sítě **4**
2.1 Tradiční architektura sítě 4
2.2 Architektura softwarově definovaných síti 5
2.3 Základní síťové prvky v SDN 7
2.4 Výhody SDN 9

3 Programování SDN sítí prostřednictvím rozhraní **11**
3.1 Jižní programovací rozhraní 11
3.2 Severní programovací rozhraní 14

4 Detekční mechanizmy **17**
4.1 Detekční mechanizmy a možnosti jich použiti 17
4.2 Detekční mechanizmus pro řízení sítě s využitím prostředků OS 19

5 Směrování **21**
5.1 Směrování v tradičních sítích 21
5.2 Směrování SDN 23
5.3 Návrh aplikace 25

6 Implementace **26**
6.1 Detekční mechanizmus 26
6.2 Řídící aplikace pro směrování dat 27

7 Testování **33**
7.1 Test 1 34
7.2 Test 2 35
7.3 Test 3 35

8 Závěr **38**

Kapitola 1

Úvod

Jednou z charakteristických vlastností moderní společnosti je využití počítačů ve všech sférách lidského života. Počítače už delší dobu obsazují místo našich permanentních pomocníků ve vyhledávání a vyměňovaní jakýchkoli informací. Takovou možnost využití počítačů nám poskytují počítačové sítě, které tvoří základ komunikace pro celý svět. Slouží nám k dorozumívání na velké vzdálenosti a k přenosu informací. Současný obrovský rozmach počítačových sítí vyžaduje jejich maximální soulad s rostoucími požadavky (větší šířka přenosového pásma, rychlost komunikace, odezva sítě, bezpečnost, správa a údržba).

Neustálé zvyšování nároků na počítačové sítě se stalo důvodem hledání nových řešení a pohledů na využívání síťové infrastruktury. Důležitým požadavkem současných sítí je zlepšení míry spokojenosti uživatelů s kvalitou poskytovaných služeb (Quality of Experience - QoE). V současné době téměř kterákoli aplikace využívá počítačové sítě pro svou plnou a konkurenční funkcionalitu (např. Skype, Seafile, TeamViewer atd.). Každá aplikace je odlišná v tom k čemu slouží a jak funguje. To definuje specifické požadavky aplikací na síť a to velkou šířku pásma, nízké zpoždění nebo obojí naráz. Vyhovění síťovým nárokům aplikaci je jednou zmožností zlepšení QoE.

Na začátku 20. století začaly se objevovat tzv. sítě nové generace (Next Generation Networks - NGN), které se snaží zaměřovat na efektivní využití, automatizaci síťových zařízení, inteligentní směrování dat a další aspekty, které by pomohly vyhovět aktuálním požadavkům. Jedním z představitelů sítě nové generace jsou Softwarově definované sítě (SDN). Jejich princip je založen na nové architektuře oddělující řídící a datovou vrstvu sítě. Podobná separace umožňuje větší přehled, kontrolu, správu a údržbu síťových infrastruktur.

Jednou z pozitivních a vylepšených vlastností SDN je směrování, které se provádí na základě tabulky toků. Tabulka toků obsahuje dodatečné informace pro zefektivnění směrování a spravuje se speciálním síťovým prvkem - kontrolerem. Výhodou takového směrování je použití nejen cílové adresy a portu pro doručení dat příjemci, ale i jiná kritéria. SDN směrování umožňuje směrovat data v síti zohledem na požadavky uživatelských aplikaci. Například při existenci rychlé cesty nebo cesty s velkou šířkou pásma by nám takové směrování umožnilo vybrat tu nejvhodnější cestu pro předávání dat konkrétního typu aplikace, což by určitě mělo pozitivní vliv na kvalitu poskytovaných služeb uživateli.

Hlavním cílem této práce bylo navrhnout algoritmus směrování v sítích SDN, který zohledňuje požadavky aplikací v sítích. Proto byly aplikace rozděleny podle dvou kritérií, a to šířky pásma a zpoždění. Součástí práce bylo vytvoření detekčního mechanizmu na koncovém zařízení pro zjištění, kterým aplikacím patří síťové toky v síti.

Práce je rozdělená do 8 kapitol. Kapitola 2 vysvětluje základní principy architektury softwarově definovaných sítí a také principy tradiční architektury. Kapitola 3 se zabývá

programováním SDN sítí prostřednictvím severního a jižního rozhraní kontroleru. V kapitole 4 je popsána detekce aplikací využívajících síťové toky pomocí systémové utility instalované na koncových stanicích. Kapitola 5 se věnuje směrování jak v tradičních sítích, tak i v sítích SDN. Popisuje také návrh aplikace běžící na kontroleru, která řídí směrování síťových toků s ohledem na požadavky síťových aplikací. 6. kapitola popisuje propojení detekčního skriptu pro koncové stanice a implementaci aplikace pro řízení sítě běžící na kontroleru. Kapitola 7 se zabývá testováním výsledného softwaru a jeho výsledky.

Kapitola 2

Softwarově definované sítě

Softwarově definované sítě (SDN z angl. Software-Defined Networking) představují novou architekturu počítačových sítí, která má řídící část sítě oddělenou od datové [1]. Součástí této kapitoly je popis tradiční architektury sítě a SDN architektury. Rozebrány jsou výhody nové architektury, které nám přinesl tento nový pohled na sítě.

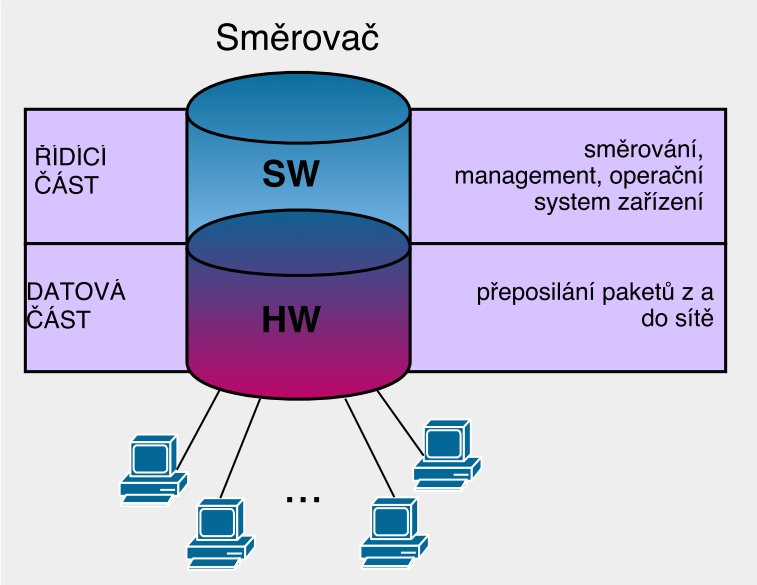
2.1 Tradiční architektura sítě

Tradiční architektura je nejvíce rozšířená a stále se používá jako hlavní. Každé síťové zařízení (směrovač, přepínač) se skládá zdatové a řídící části:

- **Datová část (data plane)** Slouží k přeposílání paketů z jednoho rozhraní na jiné. Tato část je schopna vykonávat také jednoduché funkce: validace kontrolního součtu dat (Checksum), modifikaci TTL (Time To Live), nebo zahození paketu [13]. Operace datové části se provádí nad každým paketem, což vyžaduje vysokou výkonnost v reálném čase. Z toho důvodu je datová část typicky implementována v hardwaru[2].
- **Řídící část (controle plane)** je softwarová část zahrnující operační systém pro správu sítě a řídící logiku jejího chování [13]. Na rozdíl od datové části nejsou funkce řídící části prováděny nad každým paketem a nejsou tak striktně omezeny časem jak funkce datové vrstvy, proto jsou implementovány softwarově [2].

Obě části, jak je vidět na obrázku 2.1, se nachází na jednom síťovém zařízení a fyzicky nejsou od sebe oddělitelné. Síťová zařízení v klasické architektuře jsou závislá na operačním systému, který je dán výrobcem. Podobné systémy jsou obvykle uzavřené a podporují pouze funkcionalitu, kterou výrobce pevně specifikoval. Vzhledem k vysoké integraci obou částí je nelze vyvíjet nezávisle, protože přidání funkcionality jedné části vyžaduje buď aktualizaci, nebo celkovou změnu části druhé. Takový stav činí síťové prvky uzavřené kinovacím. Z toho vyplývá, že nemáme možnost přímo přidat nebo změnit funkcionalitu klasického síťového zařízení, ale musíme si vystačit pouze s funkcionalitou nabízenou výrobcem.

V případě, že je potřeba přidat nějakou funkcionalitu, současná architektura nám umožňuje jen dvě řešení. První je samostatně ji naimplementovat a čekat na schválení standardizačními institucemi, což většinou není jednoduché, vyžaduje hodně času a vůbec negarantuje pozitivní výsledek. Druhé řešení je, že danou inovaci, často za vysokou finanční kompenzaci, naimplementuje nějaký výrobce (např. Cisco). Zavedení nové funkcionality je tak hodně časově a finančně náročné a prakticky nedostupné pro většinu správců sítí. Proto



Obrázek 2.1: Architektura klasického síťového zařízení.

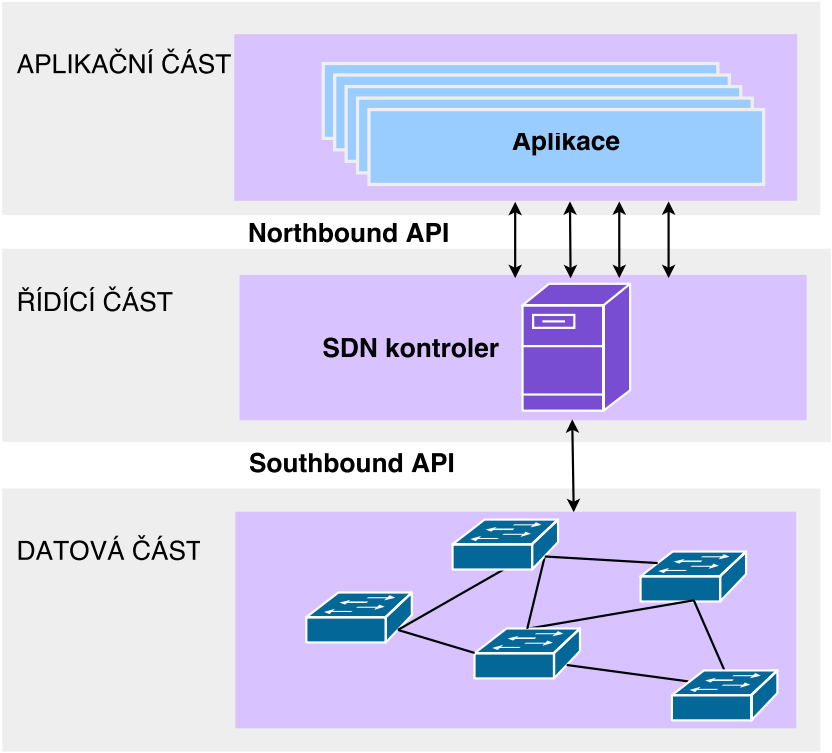
zavádění nových protokolů a služeb (a dokonce i nové verze stávajících protokolů) není vhodné [4].

Konfigurace klasických síťových zařízení probíhá prostřednictvím zabezpečeného komunikačního protokolu SSH (z angl. Secure Shell). Správce sítě se pro daný účel musí připojovat na každé zařízení zvlášť, což může být poměrně náročné vpřípadě konfigurace sítě s větším počtem síťových zařízení. Samotná konfigurace je iterativní zadání konfiguračních příkazů vrámci každého síťového prvku. Nevýhodou konfiguračních příkazů je, že se liší nejen od výrobce k výrobci, ale i v rámci jednoho výrobce (např. dvě různé verze přepínačů od firmy Cisco).

2.2 Architektura softwarově definovaných síti

Během použití a s rostoucími požadavky síťových administrátorů a aplikací se stále zjišťuje, jaké má tradiční architektura nevýhody. Softwarově definované sítě představují architekturu, která se snaží vyřešit problémy tradiční architektury. Architektura SDN se skládá ze tří částí: datové, řídící a aplikační. Každá část je spojená snásledující specifickým rozhraním, tato budou popsána v části 2.3.1. Hlavním rysem nové architektury, která je zobrazena na obrázku 2.2, je separace řídící části v softwaru od datové v hardwaru. Hlavní části architektury SDN jsou definované jako:

- **Datová část (data plane)** je vrstva, která má stejnou funkcionalitu, ale jinou architekturu, než datová část v tradiční architektuře. Z hlediska architektury může být SDN přepínač jednoho z dvou typu: klasický SDN přepínač nebo hybridní přepínač. Klasický softwarově definovaný přepínač podporuje architekturu SDN. Výhoda takového přepínače je v tom, že přepínač podporující jen jednu architekturu je jednodušší a levnější. Hybridní přepínač podporuje jak tradiční architekturu sítě, tak i SDN architekturu. Existují dvě možnosti, jak se toho dá dosáhnout.



Obrázek 2.2: Architektura softwarovì denovaných sítí.

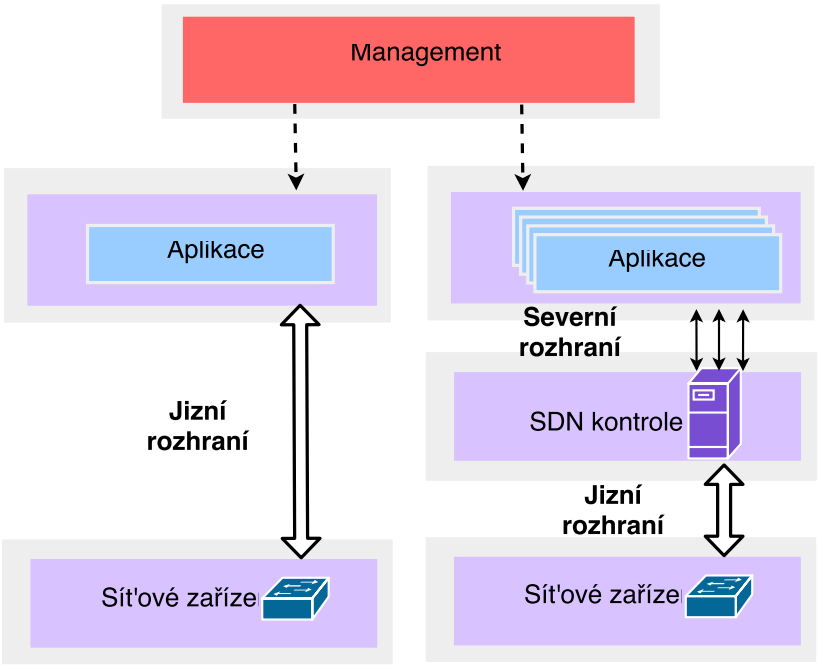
  - První varianta je rozdělení portů přepínače na porty s SDN funkcionalitou a porty s funkcionalitou, kterou mají přepínače v klasické architektuře. Rozdělení architektur podle portů je z hlediska hardwaru jednodušší možnost.
  - Druhou variantou výběru architektury je podle libovolného atributu nebo množiny atributů zpracovávaného paketu (např. IP adresy, porty, síťový identifikátor). Výběr architektury podle atributů paketu poskytuje řídící části, resp. správci sítě, větší kontrolu přeposílacího procesu (např. všechny pakety s cílovým portem 80 se zpracují jako v klasické architektuře). Na druhou stranu ne každý hardware tuto variantu podporuje.

- **Řídící část (controle plane)** má možnost být přemístěna na jiné zařízení nazývaného kontroler, což ale není nutné. Přemístění řídící části na jiné zařízení umožňuje centralizaci řízení, kdy několik přepínačů má společné řízení na nějakém serveru. Je to výhoda, kterou můžeme, ale nemusíme využít. Kontroler svými rozhodnutími definuje chování buď celé sítě, nebo jenom té části, za kterou je zodpovědný. Řídící část poskytuje pro aplikační vrstvu abstraktní pohled na síť [11].

- **Aplikační část (application plane)** je vrstva, která se obvykle nachází nad řídící vrstvou a je tvořena různými aplikacemi. Vzhledem k separaci řídící a datové části je možné tvořit aplikace pro lepší správu sítě (např. bezpečnost nebo sledování síťového provozu). Aplikace dané vrstvy řídí chování sítě přes kontroler, se kterým komunikuje přes speciální rozhraní [17, 18].

2.3 Základní síťové prvky v SDN

V této části budou popsány základní síťové prvky softwarově definovaných sítí. Tři základní elementy v SDN jsou kontroler, síťové zařízení a komunikační kanál, tyto jsou znázorněny na obrázku 2.3.



Obrázek 2.3: Architektura programovacích rozhraní v SDN sítích

Síťové zařízení

V SDN je síťové zařízení komponenta, která komunikuje s kontrolerem, přijímá jeho příkazy a na základě toho manipuluje s pakety. Takové zařízení je abstrahováno od specifičnosti směrovače, přepínače, load-balancera nebo firewallu. V SDN se tomuto zařízení říká SDN přepínač, který v hardwaru implementuje rychlé přeposílání paketů a v softwaru komunikaci s kontrolerem. SDN přepínač je možné nakonfigurovat tak, aby mohl samostatně dělat rozhodnutí a reagovat na případný výpadek sítě nebo dodatečně mohl poskytovat takové funkce jako prevence smyček (protokol STP) nebo zpracovaní protokolu.

SDN kontroler

SDN kontroler se fyzicky nachází mezi aplikační a datovou vrstvou sítě a představuje prostředníka mezi uživatelem a síťovou infrastrukturou. Je to software, který provádí řízení jednotlivých datových částí. Může být implementován jako několik programových částí, které si udržují synchronizovaný a konzistentní pohled na stav a topologii celé sítě. Jeho hlavními úlohami jsou konfigurace zařízení, zjišťování topologie, směrování, správa sítě a případná synchronizace s ostatními kontrolery.

V SDN se často používají centralizované kontrolery, což řeší řadu problémů, které jsou charakteristické pro distribuované řídící části:

- Odezva linky nebo síťového uzlu po selhání je rychlejší, protože informace o selhání nemusí být předána přes násobné množství dalších uzlů.
- Vyvarování se smyčkám by mělo být jednodušší, jelikož kontroler má úplný přehled o celé síti.

Použití centralizovaného kontroleru má nejen své výhody, ale také nevýhody:

- Každý jeden kontroler představuje jeden bod, který může způsobit selhání celé sítě. Tudíž musí existovat možnost četné redundance a synchronizace ostatních kontrolerů v případě výpadku některého z nich.
- Centralizovaný kontroler je lákavý cíl pro útok - jestliže útočník získá kontrolu nad kontrolerem, má plný přístup k síti.
- Aktualizace směrovacích pravidel přepínače z kontroleru může způsobit případ dočasné nekonzistence směrování a následné vytvoření mikrosmyček. Geografická vzdálenost důsledky tohoto problému ještě zhoršuje.

Hlavním cílem všech SDN kontrolerů je umožnit uživatelům nebo síťovým administrátorům psát vlastní aplikace, které používají kontroler buď jako prostředníka, nebo jako vrstvu abstrakce mezi síťovými aplikacemi a síťovými zařízeními.

Komunikační kanál

Komunikační kanál je rozhraní, které propojuje síťové zařízení a kontroler. Slouží pro komunikaci mezi těmito dvěma komponentami a přeposílání řídících a autentizačních dat a paketů mezi nimi.

2.3.1 Rozhraní v architektuře SDN

Veškerá komunikace řídící vrstvy s ostatními částmi SDN architektury probíhá prostřednictvím speciálních rozhraní. Rozhraní spojující aplikační část s kontrolerem se nazývá severní. Jižní rozhraní spojuje kontroler s datovou vrstvou. Podrobnější popis obou rozhraní z programovacího hlediska bude uveden vdalší kapitole.

Jižní rozhraní (southbound interface)

Hlavní úlohou jižního programovacího rozhraní je umožnit komunikaci mezi SDN kontrolerem a síťovými uzly (fyzické i virtuální přepínače), aby kontroler mohl zjistit topologii sítě, definovat síťové toky a realizovat přenos požadavků přes severní programovací rozhraní. Jižní rozhraní usnadňuje a zefektivňuje kontrolu nad sítí a umožňuje kontroleru dynamicky provádět změny v síti za běhu podle aktuálních požadavků a potřeb [14]. Toto rozhraní má svoje problémy a omezení, mezi které patří [5]:

- nízká úroveň abstrakce - programování složitějších funkcionalit sítě je velmi náročné;
- náročné pro vykonávání několika nezávislých úloh současně (QoS, směrování a load-balancer);

- souběžnost - operace SouthBound API se musí provádět ve správném pořadí.

Jižní rozhraní může být jak proprietární (OnePK), tak i standardizované (OpenFlow). V současné době je protokol OpenFlow průmyslovým standardem pro jižní rozhraní, který definuje způsob komunikace SDN kontroleru a datové části architektury [15].

Severní rozhraní (northbound interface)

Severní programovací rozhraní vytváří kontroler pro zjednodušení tvorby aplikaci pro správu sítě a také implementaci složitějších síťových funkcí. Prostřednictvím kontroleru se zapouzdřuje nízká úroveň příkazů jižního rozhraní. Existence daného rozhraní a schopnost kontroleru abstrahovat a normalizovat části SDN architektury, umožnila i běžným programátorům rychlou úpravu nebo přizpůsobení chování sítě zadaným požadavkům prostřednictvím vyšších programovacích jazyků (např. Javy, Pythonu, Ruby) [8].

Prostřednictvím severního rozhraní je mnohem jednodušší implementovat síťové funkce, např. výpočet cesty, vyhýbání se smyčkám, směrování, zabezpečení a mnoho dalších úkolů. Dané rozhraní umožňují také integrace systémů, jakými jsou OpenStack Quantum nebo VMware vCloud Director pro správu síťových služeb v cloudu [11, 14].

Severní rozhraní oproti jižnímu není standardizováno, a tudíž v současné době je dostupných přes 20 různých SDN kontrolerů, přičemž každý nabízí svoje vlastní proprietární rozhraní. Variace daného rozhraní je pro začátek třeba prozkoumat a zjistit, jaké jsou příčiny jejich vzniku a odlišnosti. SDN sítě jsou nové, takže momentálně se pouze zkoumá, jaké výhody a nevýhody nám severní rozhraní může nabídnout. Proto je většina předních síťově zaměřených organizací pevně přesvědčená, že na vedení diskuzí ohledně standardizace severního rozhraní je ještě poměrně brzy [8].

2.4 Výhody SDN

Softwarově definované sítě nám přinesly celou řadu výhod oproti architektuře tradičních sítí. Navzdory veškerým výhodám je třeba poznamenat, že funkcionalita se nijak nenavyšuje a ani se neřeší problémy spojené s přenášením dat, jako, například, problémy s TCP/IP síťovým modelem. Mezi hlavní výhody sítí SDN patři[9]:

- **Pohodlná konfigurace obsáhlé sítě** - přítomnost kontroleru nám dovoluje nakonfigurovat nebo modifikovat poměrně obsáhlou datovou síť pomocí úpravy či konfigurace chování jednoho centralizovaného kontroleru. Z toho vyplývá, že jeden síťový administrátor stihne udělat více práce. Tím pádem stačí menší množství administrátorů. Umožní to větší úsporu financí při správě sítě.

- **Komplexní pohled na síť** - kontroler má přehled o topologii celé sítě, za kterou je zodpovědný. Díky této vlastnosti může lépe předvídat možné konfliktní situace v síti i rychleji a kvalitněji řešit problémy, které již nastaly. Tím pádem se snižuje možnost vzniku mikrosmyček a černých děr, a také se redukuje jejich počet. Pro udržení provozu odpadává potřeba vytvoření a použití takových algoritmů pro obcházení, jako je Fast Reroute (FRR) nebo Loop-Free Alternates (LFA), během obnovení řídící vrstvy linky nebo uzlu po selhání.

- **Redukce a výhodné nahrazování síťových prvků** - s mnohem efektivnější řídící logikou síťových zařízení je možné zredukovat počet a typ samotných zařízení. Například pro potřeby použití firewallu nebude nutné samotné zařízení kupovat. Bude

možné si vystačit pouze s SDN přepínačem, který po vhodném naprogramování bude schopný pracovat jako firewall. Tím pádem dokážeme nahradit i jiné síťové zařízení přepínačem SDN. Další výhodou je možnost ušetřit náklady na elektřinu, například vypnutím redundantních přepínačů během nízkého zatížení (například v noci).

- **Snazší testování nových funkcí** - přepínač je možné rozdělit na několik částí, kde každá bude využívat jinou část jeho datové vrstvy (slicing). Například přidáme nový modul do programu řídící vrstvy kontroleru, který bude s jednou polovinou portů na přepínači pracovat jinak než s druhou. Takové rozdělení nám dává možnost otestovat nové funkce uvnitř produkční sítě tím, že se pro testování využijí pouze vybrané porty (například nový směrovací algoritmus). Další výhodou je postupné zavádění nové funkcionality postupným přesunem provozu z jedné oddělené části do dalších.

- **Snížení nákladů na síťové zařízení** - separace řídící části (softwarové) od datové (hardwarové) snížila složitost síťových zařízení a jejich výrobní náklady. Už se nemusí provádět drahý návrh, implementace a testování obou částí současně při modifikaci jedné z nich.

- **Nezávislost na výrobci** - pomocí speciálního rozhraní mezi kontrolerem a síťovým zařízením SDN sjednotilo různé typy operačních systémů a rozhraní pro konfiguraci zařízení. Tento krok odstranil závislost na výrobcích a standardizačních institucí, což dovolilo rychlejší inovace.

- **Lepší virtualizace** - SDN lépe virtualizuje síť. Kontroler je schopný abstrahovat přepínače, čímž pro aplikace dokáže celou síť zobrazit jako jeden virtuální, obrovský přepínač. V této situaci řídící aplikace nemusí brát v úvahu podrobnosti topologie, což znamená, že nám virtualizace sítí povoluje fyzickou změnu topologie beze změn pro aplikace.

- **Zvýšená spolehlivost** - přímá interakce člověka se sítí je často hlavním důvodem síťových výpadků. Také pravděpodobnost konfiguračních chyb narůstá přímo s růstem počtu zařízení. Kontroler automatizuje některé z těchto interakcí a řeší konzistenci konfigurace. Z toho vyplývá, že SDN je schopné redukovat počet síťových politik a procedur, které slouží pro interakci člověka se sítí. Taková redukce má pozitivní vliv na spolehlivost a provozní náklady.

Kapitola 3

Programování SDN sítí prostřednictvím rozhraní

Síťová zařízení jsou konfigurována kontrolerem přes programové jižní programovací rozhraní. Jak již bylo zmíněno dříve, nad tímto rozhraním může být implementován kontroler, který umožňuje spravovat zařízení přes severní rozhraní. V rámci SDN se využívají jižní a severní rozhraní. V této kapitole budou tato rozhraní popsána podrobněji z hlediska programátora. Součástí kapitoly je popis kontroleru POX, který se v této práci používá.

3.1 Jižní programovací rozhraní

Jižní rozhraní se dá rozdělit na dvě části. První část popisuje komunikační rozhraní mezi přepínačem a kontrolerem (např. formu dat). Druhá část popisuje vnitřní strukturu přepínače pomoci speciálního protokolu (např. OpenFlow, NetConf, OnePK od firmy CISCO), nebo teoreticky si dokážeme vystačit i s přístupem ke konzoli pomocí telnetu nebo SSH.

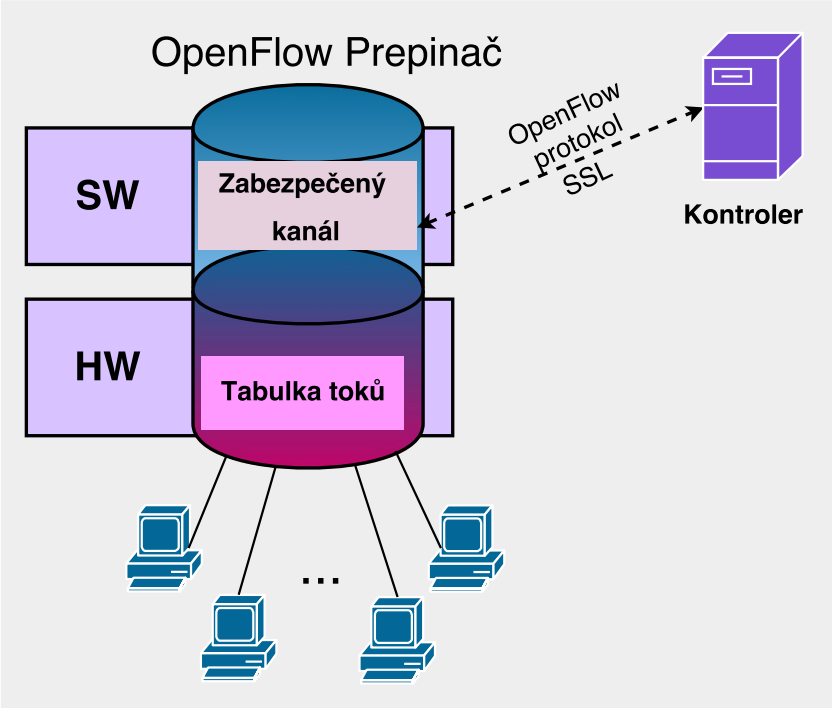
3.1.1 OpenFlow

OpenFlow (OF) je první a nejznámější standard jižního SDN rozhraní, který je veden organizaci Open Networking Foundation. Jedná se o souhrn protokolů, programových rozhraní a jeden z možných pohledů na softwarově definované sítě. Aktuálně nejnovější verze protokolu je 1.5.1. Architektura OpenFlow je znázorněna na obrázku 3.1.

Přepínač, který protokol OpenFlow podporuje se nazývá OpenFlow přepínač. Prostřednictvím tohoto rozhraní, SDN kontroler propaguje změny, určené správcem sítí, dolů do tabulky toků přepínače. Daná funkce umožňuje správci sítě dělení provozu či kontrolních toků pro optimalizace výkonu, lepší a lehčí testování nových konfiguraci a aplikací[16].

OpenFlow definuje:

- množinu instrukcí, které kontrolují chování přeposílání na přepínači;
- množinu zpráv, které přepínač může posílat kontroleru pro informování o změnách, které by mohly ovlivnit přesměrování;
- formát pro uložení pravidel na přepínači;
- protokol pro odesílání a přijímání zpráv mezi kontrolerem a přepínačem.



Obrázek 3.1: Architektura OpenFlow.

3.1.2 Protokoly v OpenFlow

Protokoly v OpenFlow se dělí na síťový protokol (OpenFlow) a konfigurační protokol (OpenFlow-Config). Oba dva tyto protokoly byli vyvinuty organizací Open Networking Foundation.

Konfigurační protokol

OpenFlow-Config je protokol, používaný pro konfiguraci síťových zařízení. Nemění část, která se zabývá přeposíláním paketů (datová část). Příkladem použití protokolu je nastavování IP adres a portů rozhraní nebo zapnutí/vypnutí vybraného rozhraní.

Síťový protokol

Tato část OpenFlow protokolu se využívá pro rychlou změnu stavu síťového zařízení. Jeho úlohou je komunikace mezi kontrolerem a přepínačem a následná modifikace datové části zařízení. OpenFlow přepínač využívá koncept datových toků k identifikaci síťového provozu na základě kontrolerem definovaných pravidel, která z něj byly naprogramována. Přepínač se skládá z následujících položek:

- Rozhraní zabezpečeného komunikačního kanálu, které odpovídá definici jižního rozhraní;
- Tabulky toků, která obsahuje soubor záznamů o datových tocích.

3.1.3 Tabulka toků

Tabulka toků (Flow table) je jednou ze základních struktur OpenFlow přepínače (obrázek 3.2). Obsahuje záznamy datových toků, které se skládají z polí hlaviček, množiny akcí, a počítadel pro statistiky[9].

Vyhledávaní v tabulce toků probíhá shora dolů pravidly podle priorit (od největší po nejmenší). Když se během vyhledávání najde nějaké pravidlo (záznam), které se ve všech položkách shoduje s porovnávaným záznamem, vyhledávání se považuje za úspěšné a skončí. Podle potřeby může být na každou z položek pravidla aplikován zástupný znak typu „*“(libovolná hodnota), který umožňuje agregace toků.

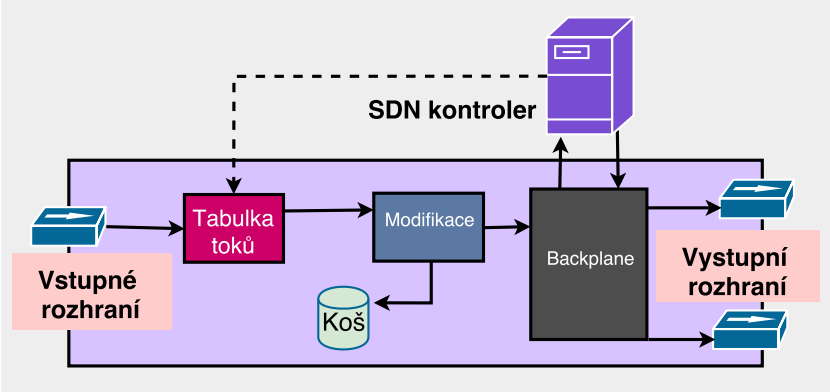
Priorita	Vstup. port	MAC src	MAC dst	Eth typ	VLAN ID	IP src	IP dst	IP Prot	TCP sport	TCP dport	Akce	Počítadla	
												packetů	bajtů
54796	*	*	*	0800	53	*	192.10.0.1	4	*	80	port 1	42	5847
4795	*	*	*	0800	*	*	192.10.0.1	4	*	80	port 2	738	35755
4794	*	*	*	0800	*	*	192.10.0.1	4	*	*	port 3	2345	162748
5	···	···	0800	83	···	10.10.10.1	4	9467	80				

Obrázek 3.2: Tabulka tokú v OpenFlow přepínačích.

- **Pole hlaviček (Header fields)** je využíváno pro porovnání záznamu s příchozími pakety v tabulce toků. Ve verzi OpenFlow 1.0 je datový tok 10-tice, která se skládá ze vstupního portu, VLAN ID, zdrojové a cílové MAC adresy, typu ethernetového rámce, zdrojové a cílové IP adresy, IP protokolu, zdrojového a cílového TCP/UDP portu. Kromě možnosti použiti zástupného znaku typu „*“ pro agregaci jakéhokoliv políčka je možné zadat rozsah adres (adresu sítě) pro IP adresy.
- **Počítadla (Counters)** - do této části tabulky se ukládají statistiky o toku, které se aktualizují s každým přijatým paketem, pro který byl nalezen záznam v tabulce toků.
- **Akce (Actions/Instructions)** se využívají pro definování chování přepínače pro pakety, pro které byl nalezen záznam v tabulce toků. OpenFlow definuje povinné akce: přeposílání a zahození. Při vykonání akce přeposílání je nutně specifikovat fyzický, virtuální nebo některý z rezervovaných portů:
  - **ALL** - rozeslání paketu na všechny porty kromě příchozího;
  - **CONTROLLER** - odeslání paketu na kontroler;
  - **LOCAL** - odeslání paketu na stack přepínače pro zpracování operačním systémem (pouze u hybridních přepínačů).
  - **TABLE** - provedení instrukce z tabulky pravidel;
  - **IN_PORT** - odeslání paketu zpět na vstupní port.

Dále OF definuje volitelné akce: zařazení do fronty, modifikace polí (možnost modifikovat jakoukoli položku, podle které se dále porovnává vrámci tabulky toků), snížení, resp. zvýšení TTL nebo vložení, resp. odstranění tagu (VLAN, MPLS atd.).

V podstatě tabulka toků pracuje následovně (obrázek 3.3): Pro všechny přijaté pakety se vyhledává příslušný záznam v tabulce toků. Pokud je v tabulce odpovídající záznam nalezen, přepínač provede všechny akce, které jsou pro daný tok specifikovány. Pokud pro paket nebyla nalezena shoda, je přeposlán přes zabezpečený kanál do kontroleru. Kontroler pak pomocí přidávání, upravování a odstraňování záznamů v tabulce toků rozhodne, jak se budou takové pakety zpracovávat.



Obrázek 3.3: Funkce tabulky toků.

3.2 Severní programovací rozhraní

Severní aplikační rozhraní je programovací rozhraní, které komunikuje s kontrolerem a je prostředníkem mezi ním a aplikační části. Aplikace využívají severní programovací rozhraní pro konfiguraci kontroleru. Kontroler na základě konfigurace, kterou mu posílá aplikační část, řídí a modifikuje síť. Z hlediska programování toto rozhraní abstrahuje nízkou úroveň instrukcí jižního rozhraní a samotný kontroler, čímž umožňuje programování složitějších síťových funkcí[9].

3.2.1 Možnosti instalování pravidel do tabulky toků

Aplikační část definuje síťovou politiku kontroleru na základě aktuálního stavu a topologie sítě přes severní rozhraní. Následovně převádí tuto politiku na sadu OpenFlow pravidel, které prostřednictvím jižního programovacího rozhraní instaluje do tabulky toků síťových zařízení. Opačný tok dat začíná odesíláním události zjižního rozhraní na kontroler, kde se zpracovávají, a následně jsou přístupné aplikacím přes severní rozhraní. Existují dvě možnosti instalování pravidel do tabulky toků, a to reaktivní a proaktivní.

Reaktivní způsob

Reaktivní způsob je založen na tom, že neznámý paket, který přijde do jakéhokoliv přepínače se přeposílá na kontroler. Kontroler vytvoří požadované pravidlo pro daný tok a pošle ho do tabulek toků každého přepínače v síti. Vzhledem k tomu, že všechny neznámé pakety se posílají na kontroler, získává kontroler detailnější přehled o dění na síti, které je možné využít pro řízení sítě. Nevýhodou reaktivního chování je zpoždění vzniklé odesláním neznámých paketů kontroleru, vytvořením příslušných pravidel a jejich nainstalování na síťová zařízení. Až po nainstalování pravidel jsou všechny další pakety rovněž odeslány na kontroler. S rostoucím počtem odesílaných paketů na kontroler se vytížení přepínače z důvodu zatížení komunikačního kanálu zvyšuje.

Proaktivní způsob

Oproti reaktivnímu chování je proaktivní založeno na tom, že kontroler je předem informován o očekávaných tocích. Z informací o nich kontroler předinstaluje potřebná pravidla do tabulky toků přepínačů. V případě příchodu neznámého toku budou jeho pakety zahozeny kontrolerem.

Výhodou proaktivního způsobu je snížení počtu přenášených paketů na kontroler, přičemž je možné přeposílání paketů na kontroler úplně eliminovat. Vzhledem k tomu, že všechna pravidla se již nachází na síťových prvcích, zpoždění způsobené instalací pravidla nevzniká. Nevýhodou je složité zjištění předem, jaké toky mají být nainstalovány, což v některých aplikacích nemusí být možné. Z důvodu omezené kapacity tabulky toků jsou využívána agregační pravidla, čímž se zmenšují možnosti monitorování sítě podle statistik jednotlivých pravidel.

3.2.2 Kontroler POX

Jedním z příkladů OpenFlow kontrolerů je POX[1]. POX je open source kontroler implementovaný v Pythonu, který vychází z kontroleru NOX. POX kontroler poskytuje asynchronní, událostmi řízené programové rozhraní. Obsahuje pomocné metody, frameworky a API pro interakci s dalšími SDN přepínači, debuggování a síťové virtualizace. Obsahuje znovupoužitelné komponenty pro výběr cesty nebo zjišťování topologie. Podporuje podobné GUI a virtualizační nástroje, jako NOX. Ve srovnání s ostatními kontrolery je pomalejší a méně výkonný. V současné době POX podporuje komunikační protokol OpenFlow verze 1.0[10].

Z programovacího hlediska je POX dodáván s řadou komponent implementujících základní funkcionalitu (L2 přepínač, firewall). Vývojáři dokážou rozšiřovat funkcionalitu kontroleru pomocí vytváření nových komponent [2].

[1] https://openflow.stanford.edu/display/ONL/POX+Wiki
[2] http://www.brianlinkletter.com/using-the-pox-sdn-controller/

POX komponenty (moduly)

Při spuštění kontroleru je možné specifikovat vícero komponent, které budou běžet společně. Moduly kontroleru POX mezi sebou mohou spolupracovat. Bohužel ne všechny komponenty dokážou dobře pracovat v kombinaci s dalšími komponentami nebo aplikacemi, a navíc některé moduly jsou závislé na ostatních komponentách. V případě použití několika modulů současně se kontroler musí rozhodnout, který z nich se aplikuje jako první. Řešením daného problému je priorita modulů od nejvyšší do nejnižší. Například máme kontroler, na kterém současně běží komponenty implementující směrování a firewall. Každý z těchto modulů má nastavenou prioritu, přičemž priorita komponenty firewall je vyšší. To znamená, že když dojde nový paket, jako první ho získá modul firewall. Následně se firewall rozhodne, zda je třeba daný paket přeposlat dalšímu modulu na zpracování nebo ho zahodit. Při zahození paketu jakýmkoliv modulem nejsou o přijetí paketu další moduly vůbec informovány.

Severní rozhraní POXu má formu obslužných procedur, které se spouštějí událostmi vyvolanými kontrolerem, např. připojení/odpojení přepínače, zapnutí/vypnutí portu, přijmutí paketů na kontroleru apod. V rámci každého modulu programátor implementuje vybrané procedury, které kromě logiky řízení obsahují příkazy pro odeslání OpenFlow zpráv pro konfiguraci síťových přepínačů.

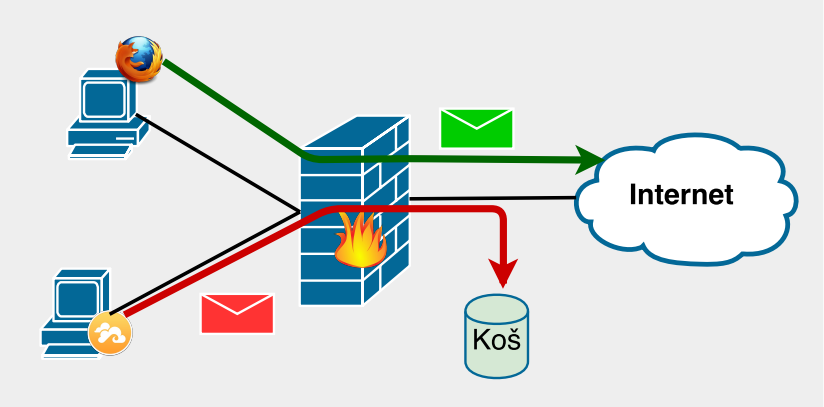
Kapitola 4

Detekční mechanizmy

V oblasti počítačových sítí existuje celá řada detekčních mechanizmů. Jejich hlavním cílem je zjištění příslušnosti paketů, resp. datových toků k aplikacím. Znalosti, které nám detekční mechanizmy poskytují, mají široké využití pro zlepšení bezpečnosti nebo řízení v počítačových sítích. Existuje vícero možností zjištění, které aplikaci patří nějaký paket síťového toku. Jednou z těchto možností je použití kombinací utilit operačního systému. V rámci této práce se informace získané navrhnutým detekčním mechanizmem použijí pro zlepšení směrování v SDN sítí.

4.1 Detekční mechanizmy a možnosti jich použiti

Detekční mechanizmy slouží hlavně k mapování paketů na aplikace, které je zpracovávají. Další využití je zjišťování, co je to za počítač (identifikace, IP a MAC adresy) a jaké je jeho vytížení. Na základě dané informace chceme v rámci této práce zefektivnit řízení sítě, konkrétně směrování. V takovém případě by každé síťové zařízení v rámci SDN sítě vědělo, jaké aplikace jsou zodpovědné za přicházející pakety, a jak se s nimi mají zacházet. Získané znalosti se využívají například pro účtování mobilních datových tarifů, odchytávání dat určitého toku ze sítě (zákonné odposlechy) atd.



Obrázek 4.1: Firemní firewall, který zahazuje pakety aplikace Seafile, směřujících do internetu a propouští pakety aplikace Firefox.

Příklad využití znalostí aplikací při řízení sítě je zobrazen na obrázku 4.1. Obrázek znázorňuje firemní firewall, který je schopen podle paketu zjistit název aplikace zpracovávající

příslušný síťový tok a na základě toho se rozhodnout, zda paket zahodí nebo pošle dále po síti. Obrázek znázorňuje příklad, kdy by aplikace pro zálohování firemních dat (Seafile) neměla mít možnost přenášet data z, resp. do, internetu a tyto pakety jsou zahozeny. Na druhou stranu webový prohlížeč (Firefox) by přístup k internetu mít měl.

4.1.1 Současné možnosti detekce

Existuje několik způsobů, přiřazování paketů k aplikacím. Všechny způsoby lze rozdělit do dvou kategorií: detekce na koncových zařízeních a detekce v rámci sítě. Následuje popis možnosti detekce spolu s jejich výhodami a nevýhodami.

Porty

Síťový administrátor definuje, na kterých portech běží konkrétní aplikace. Na základě daného nastavení budou pracovat síťová zařízení (firewall, přepínač, kontroler). Například podle nastaveni administrátora bude Firefox využívat port 80. Pro síťové zařízení to znamená, že libovolný paket s cílovým portem 80 patří aplikaci webový prohlížeč. Výhodou dané možnosti detekce je nenáročnost na implementaci a také fakt, že odpadá nutnost úpravy koncových stanic či aplikací. Nevýhodou je, že na vybraném portě mohou běžet i jiné aplikace.

Hloubková analýza paketů

Hloubková analýza paketů (Deep packet inspection) je jedna z forem inspekce paketů v počítačových sítích, která analyzuje nejen hlavičku, ale i obsah paketu při jeho průchodu kontrolním bodem (síťovým zařízením) v síti. Při hloubkové analýze paketů se zjišťuje, o jaký protokol se jedná, a na základě čeho se přiřazuje daný síťový tok k aplikaci, které pravděpodobně tok patří. Například se při analýze dat zjistí, že se jedná o protokol HTTP. Ze zjištěné informace se dá odvodit, že síťový tok, ke kterému patří prohledávaní paketů, nejpravděpodobněji patří aplikaci webový prohlížeč. Výhodou hloubkové analýzy paketů je absence nutnosti úpravy koncových stanic neboli aplikací. Další výhodou je skutečnost, že aplikace nemusí běžet na pevně daném portu, oproti předchozímu způsobu. Nevýhodou je, že hloubková analýza je náročná a nedokáže zjistit potřebné informace v případě šifrovaného provozu.

Integrace v aplikacích

V současné době existují aplikace, které mají přímo v sobě integrované ohlašovaní svých datových toků aplikacím pro správu sítě. Jedním z příkladů také aplikace je VLC media player. Výhodou řešení je, že oproti hloubkové analýze paketů je možné detekovat i šifrovaný provoz a také nezatěžuje síťová zařízení. Nevýhoda je ale v tom, že aplikace musí být upravené, což ještě neproniklo do širokého použití, a modifikace aplikací ve vlastní režii je drahá a někdy i nedostupná záležitost.

Detekce prostřednictvím operačním systému

Další možností je detekce aplikací v rámci operačního systému (OS). Taková detekce funguje na základě systémových utilit, které jsou většinou dostupné již od výrobce. Výhodou

tohoto způsobu je fakt, že dané řešení nevyžaduje úpravy aplikace, nezatěžuje síťová zařízení a funguje i pro šifrovaný provoz. Nevýhodou této detekce je nutnost mít v systému nainstalovaný modul, což u některých zařízení (např. tiskárna) může být problém.

4.2 Detekční mechanizmus pro řízení sítě s využitím prostředků OS

V rámci této práce na základě detekce prostřednictvím utilit OS byl navrhnout detekční skript, běžící na různých OS vjazyce Python. Systémové utility využité tímto detekčním mechanizmem zjišťují nová síťová spojení pomocí tabulky aktivních spojení. Tabulka aktivních spojení se kontroluje každý předem určený časový úsek (řádově sekundy). Pro nová spojení se na základě identifikačního čísla procesu vyhledávají jména aplikací. Aby skript zbytečně neohlašoval všechna spojení, byl implementován filtr. Filtr určuje názvy aplikací, jejichž spojení ohlašuje řídící aplikaci.

4.2.1 Propojení detekčního skriptu s kontrolerem

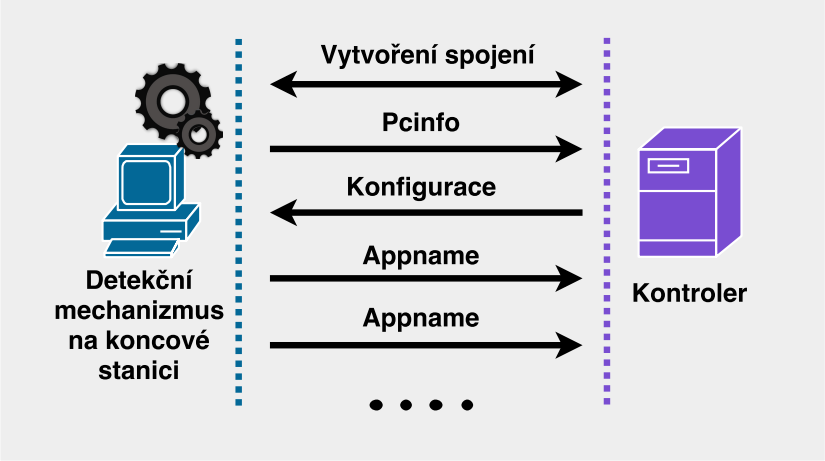
Pro aplikování detekovaných aplikací v řízení sítě je nutné propojit detekční mechanizmus s kontrolerem. K tomuto účelu byl navržen komunikační protokol, který je přenášen prostřednictvím protokolu TCP. TCP spojení je vytvořeno ihned po spuštění detekčního skriptu a je udržováno během celého procesu komunikace. Komunikační protokol obsahuje následující správy:

Pcinfo – obsahuje základní informace o koncové stanici (např. název operačního systému, jméno uživatele a počítače). Zpráva se odesílá na kontroler pouze jednou, a to při navázaní spojení. Daná funkcionalita je v rámci práce implementována pro využití možných rozšíření;

Appname – obsahuje pětici popisující nově detekovaný síťový tok spolu s názvem aplikace, která tok zpracovává.

Konfigurace – pomocí této zprávy kontroler po navázaní spojení konfiguruje detekční skript. Zpráva se skladá z položek:

- režim skriptu – položka určující režim ohlašování nově detekovaných aplikaci. V současné verzi skript podporuje pouze režim „push".
- časový interval v sekundách – používá se pouze při režimu „push". Určuje interval, ve kterém jsou nová spojení pravidelně odesílána kontroleru.
- seznam aplikací – položka obsahující seznam jmen aplikací. Pouze spojení patřící některé z těchto aplikací jsou odeslána na kontroler.



Obrázek 4.2: Režim „push“

Princip fungování režimu „push“ je zobrazen na obrázku 4.2, kde je vidět, jak probíhá komunikace kontroleru a detekčního mechanizmu. Po navázání spojení s kontrolerem detekční mechanizmus ihned odesílá zprávu Pcinfo. Kontroler po navázání spojení posílá na koncovou stanici zprávu Konfigurace. Podle přijaté konfigurace bude detekční mechanizmus posílat zprávy Appname na kontroler.

Kapitola 5

Směrování

Směrování je proces výběru nejlepší cesty od zdroje k cíli pro doručení paketů v síťovém prostředí. Přeposílání paketů od zdroji k cíli probíhá přes síťové uzly, kterými jsou například směrovač, firewall a přepínač. Tato kapitola se zabývá směrováním v počítačových sítích a popisem směrování v tradiční a SDN architektuře. Směrování v sítích SDN usnadňuje směrování podle obsahu, které je spolu s návrhem programu pro směrování popsáno na konci kapitoly.

5.1 Směrování v tradičních sítích

Směrování v tradičních sítích zajišťuje síťová vrstva modelu ISO/OSI. Na každém zařízení běží směrovací algoritmus, který prostřednictvím komunikace s jinými zařízeními naplňuje svou směrovací tabulku. Na každém síťovém zařízení směrujícím pakety běží nezávislý směrovací proces, který komunikuje s ostatními procesy pomocí výměny zpráv. Zařízení si pomocí zpráv vyměňují znalosti o topologii sítě, na základě které naplánují směrovací tabulku[6].

5.1.1 Směrovací tabulka

Směrovací tabulka je datová struktura, která se nachází v řídící části síťového zařízení. Obsahuje záznamy sítí, metriky (vzdálenosti) spojené s těmito cestami a slouží pro směrování paketů přecházejících síťovými zařízeními. Podle informací ve směrovací tabulce se systém rozhoduje, jak naložit s přijatými nebo odesílanými pakety[1].

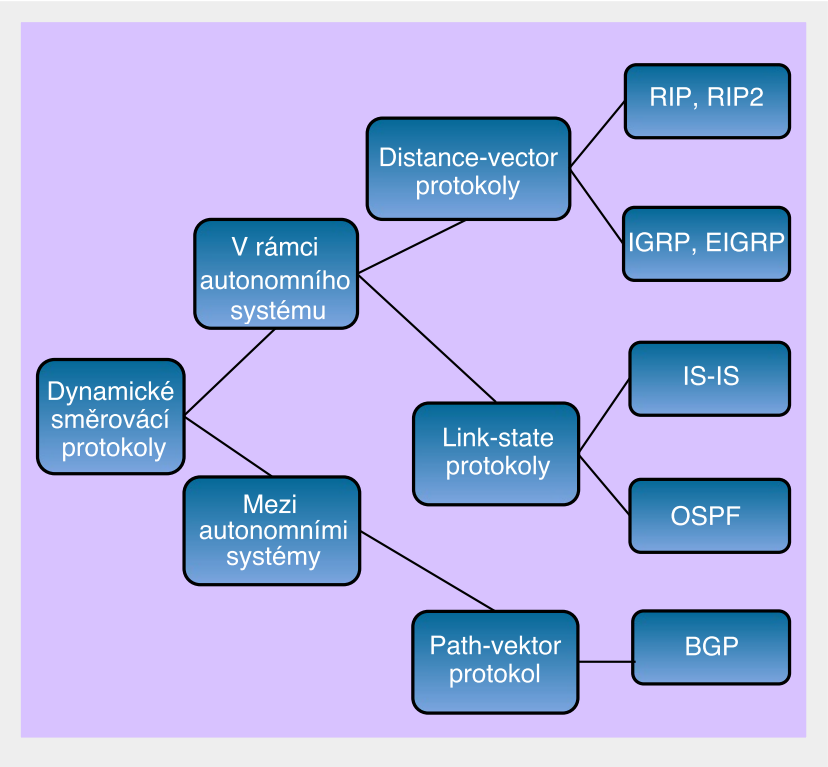
Každý paket obsahuje informace o jeho zdroji a cíli (zdrojový/cílový port, zdrojová/cílová IP adresa, zdrojová/cílová MAC adresa atd.). Když síťové zařízení přijme paket, začne jej zpracovávat. Při zpracování paketu probíhá vyhledávání shody jeho cílové adresy s položkami záznamů v směrovací tabulce podle algoritmu nejdelšího shodného prefixu (longest prefix match algoritmus). Při nalezení shody je paket zpracován, nebo je poslán dalšímu síťovému zařízení spolu s dekrementací jeho TTL nebo se zahodí [3].

Existují dva způsoby naplnění směrovacích tabulek, a to staticky a dynamicky. Statické naplnění tabulky manuálně provádí správce sítí, který jako jediný může modifikovat obsah tabulky. Nevýhodou je to, že daný způsob není použitelný při použití rozsáhlejších sítí. Dynamické naplnění směrovací tabulky probíhá prostřednictvím směrovacích protokolů, které spravují její obsah v závislosti na topologiisítě[2].

[1] http://www.faqs.org/docs/linux_network/x-087-2-issues.routing.html
[2] http://www.faqs.org/docs/linux_network/x-087-2-issues.routing.html

5.1.2 Dynamické směrovací protokoly

Směrovací protokol je proces běžící na směrovači pro určení nejvhodnější cesty, po které by se měly přeposílat pakety k jejich cílům. Procesy mezi sebou neustále komunikují pomocí výměny zpráv. Přenášené zprávy zahrnují informace o aktuální topologii sítě a zapojených sítí. Na základě zjištěné topologie směrovací protokoly vytvářejí záznamy, které mají být vložené do směrovacích tabulek síťových zařízení.



Obrázek 5.1: Rozdělení dynamických směrovacích protokolů.

Na obrázku 5.1 je znázorněno rozdělení dynamických směrovacích protokolů. Podle použití se protokoly dělí na protokoly v rámci autonomního systému a protokoly mezi více autonomními systémy. Za autonomní systém se považuje síť, která se nachází pod správou jedné organizace. Protokoly v rámci autonomního systému se dělí na dvě třídy: Link-State protokoly a Distance Vector protokoly.

- **Link-State protokoly** například OSPF, IS-IS. Protokoly si vyměňují informace o všech směrovačích a linkách v rámci své oblasti, kterou znají. Kromě toho jsou mezi oblastmi přenášena stručná data pro umožnění komunikace do vzdálenějších oblastí. Každý směrovač tak má k dispozici topologii sítě, ze které vypočítává optimální hodnoty do směrovací tabulky. Obecně jsou tyto protokoly komplexnější, vyžadují členění do oblasti a mají lepší konvergenci než distanční protokoly[7].
- **Distance Vector protokoly** například RIP, EIGRP. Obecně je tato třída protokolů výpočetně méně náročná, než třída linkových protokolů. Směrovače přijímají od sousedních směrovačů informace jak daleko (metrika) a kudy (rozhraní) je to do konkrétních částí sítí[12].

K třídě protokolů mezi autonomními systémy patří Path Vector Protocol.

- **Path Vector protokol** (např. BGP) je třída, která vznikla rozšířením třídy distančních protokolů. Základní myšlenka obou tříd je stejná až na ten rozdíl, že uzly vytvářejí směrovací tabulky a propagují je do dalších uzlů v sousedních autonomních systémech. Každý jeden takový uzel autonomního systému jedná jménem celého autonomního systému.

  Protokoly jsou určené pro velké sítě, sčímž třídy protokolů vrámci autonomního systému mají problémy. Protokol BGP poskytuje větší flexibilitu při výběru cest, avšak konvergence je velmi pomalá[3].

5.2 Směrování SDN

Klíčovým rozdílem SDN směrování od tradičního směrování je, že směrování v takových sítích řeší kontroler. Kontroler má celkový přehled o topologii, což snižuje pravděpodobnost výpadků nebo konfliktů, které mohou v síti nastat, a také umožňuje jejich operativní vyřešení. Oproti tradičnímu je SDN směrování centralizované. Centralizace poskytuje možnost vytvoření pravidel pro směrování a jejích následující distribuci do přepínačů z jednoho místa v síti. Daný SDN přístup je jednodušší, rychlejší a méně náročný na režii než tradiční přístup, kde jsou pravidla vytvářena a synchronizována každým síťovým zařízením[9].

Dalším rozdílem v SDN je použití přepínání namísto směrování paketů. Přepínání paketů probíhá na základě záznamů v tabulce toků, kterou prostřednictvím kontroleru plní a spravují aplikace. Po přijetí paketu síťovým zařízením začne jeho zpracovávání pro následné přepínání. Záznamy pro pakety jsou vyhledávané sekvenčně dle priority záznamů (viz. podsekce 3.1.3). Vzhledem k principu fungování tabulky toků je možné přepínání paketů nejen podle cílové IP adresy, ale podle libovolného atributu.

5.2.1 Směrování podle obsahu

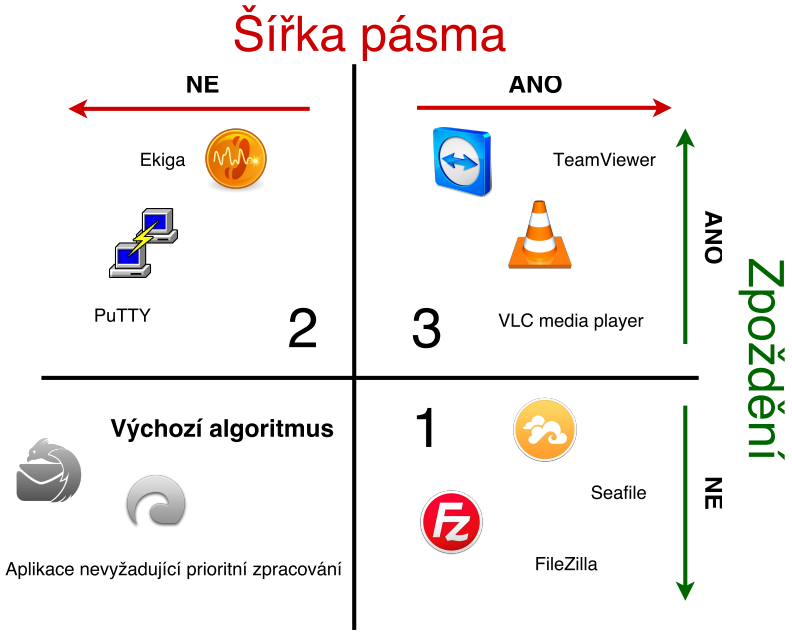
Jednou z výhod SDN směrování je, že umožňuje směrovat netradičně, například podle obsahu přenášených paketů. Způsob vytváření přepínacích pravidel definuje, podle čeho se směruje (např. při vytváření pravidel podle znalosti aplikaci, v síti se směruje podle aplikace).

Každá aplikace má svoje požadavky na přenos dat, které by při návrhu směrovacích algoritmů měly být zohledněny. Pro umožnění zohlednění daných potřeb se aplikace v rámci této práce rozdělují podle dvou kritérií. Prvním kritériem je maximální požadovaná šířka pásma. Druhým kritériem je nízké zpoždění, které se určuje počtem síťových zařízení, resp. přepínačů, mezi odesilatelem a příjemcem. Na základě těchto dvou kritérií byly navrhnuty 4 směrovací algoritmy, které jsou zobrazeny na obrázku 5.2.

5.2.2 Směrovací algoritmy

Všechny nové datové toky aplikací jsou stále směrované podle výchozích pravidel. Dané směrování trvá, dokud detekční skript nedetekuje tok od prioritní aplikace. Po nalezení se tok posílá na kontroler ve formě pětice spolu s názvem aplikace. Pětice se skládá z protokolu L4 vrstvy, zdrojové/cílové IP adresy a zdrojového/cílového portu. Kontroler na základě požadavků aplikace zvolí vhodný směrovací algoritmus pro určení nejvhodnější cesty.

[3] https://tools.ietf.org/html/rfc1322



Obrázek 5.2: Rozdělení aplikací podle jejich požadavků na přenos dat.

Pro prioritní aplikace byly navrhnuty 3 algoritmy (obrázek 5.3), které se použijí pro zjištění optimální cesty a dalšího směrování prioritních dat. Podle zvolené cesty se nahrají pravidla řídící přenos dat dané pětice na každý z přepínačů, čímž se umožní směrování podle obsahu.

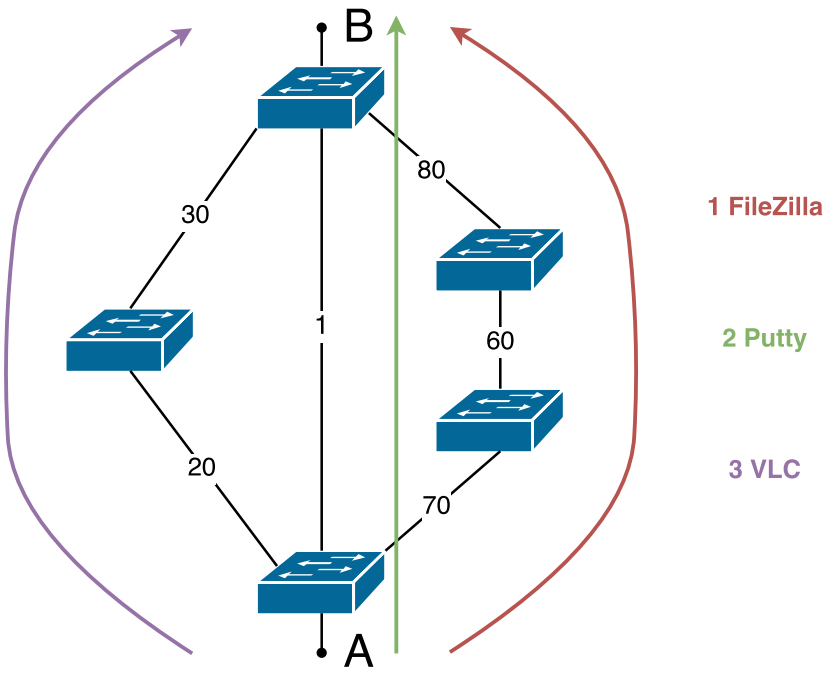
Výchozí algoritmus se používá nejen pro směrování datových toků nových spojení, ale také pro směrování dat aplikaci, které nevyžadují prioritní zpracování. Daný algoritmus funguje na zakladě generování proaktivních pravidel pro každý přepínač. Nejvyšší možná rychlost linky se zde bere bez zohlednění její aktuální zátěže. Směrování probíhá na základě cílové IP adresy paketů.

Algoritmy pro prioritní směrování:

1. **algoritmus** určený pro směrování dat aplikace požadující co největší šířku pásma. Algoritmus vyhledává cestu s největší kapacitou podle aktuální vytíženosti linek;

2. **algoritmus** určený pro směrování dat aplikacím, kterým nezaleží na šířce pásma, ale na nejrychlejší cestě. Algoritmus vyhledává nejkratší cestu k příjemci bez ohledu na maximální rychlost a vytíženost linek;

3. **algoritmus** určený pro směrování dat aplikacím, které potřebují k přenosu jak minimální zpozdění, tak určitou šířku pásma. Algoritmus hledá nejkratší cestu mezi linkami s minimální dostatečnou rychlostí pro přenos dat, kterou definuje síťový administrátor. V případě, že žádná cesta splňující tuto podmínku neexistuje, začne se hledat cesta i s linkami nevyhovujícími zadanému kritériu.



Obrázek 5.3: Příklad algoritmu pro výběru nejlepší cesty mezi odesilatelem a příjemcem dat pro různé typy prioritních aplikací.

5.3 Návrh aplikace

Celá aplikace má fungovat takovým způsobem, že detekční skripty na koncových zařízeních budou zjišťovat, jestli není nějaký nový proces, který začal běžet a potřeboval by přistupovat k síti. Když se zjistí, že se takový proces objevil, koncové zařízení pošle tuto informaci kontroleru, který podle toho zařídí směrování paketů, které nesou data dané aplikace/procesu podle toho, zda je a nebo není prioritní a jestli lépe vyhovuje pro přeposlání paketů rychlá/drahá nebo pomalá/levná linka. Seznam prioritních aplikaci bude nastavován síťovým administrátorem. Ve výsledku by aplikace zefektivnila směrování a zlepšila celkový stav sítě.

Kapitola 6

Implementace

V této kapitole bude popsána implementace praktické části bakalářské práce. Praktická část se skládá z detekčního skriptu pro koncové stanice a aplikace pro kontroler. V rámci detekčního skriptu budou popsány využité utility OS pro zjištění nových datových toků a jejich další zpracování před odesláním na kontroler. Součástí kapitoly je podrobný popis implementace výsledné směrovací aplikace, který se skládá z výčtu implementovaných událostí, směrovacích algoritmů atd.

6.1 Detekční mechanizmus

Detekční mechanizmus byl implementován jako multiplatformní skript v jazyce Python, který podporuje operační systémy Windows, GNU Linux a FreeBSD. Skript je založen na systémových utilitách, které se používají pro zjištění informaci o síťových tocích na koncové stanici.

Detekční skript se spouští s dvěma parametry, a to IP adresou a portem vzdáleného kontrolera, na TCP socket kterého, má být napojen. Po navázaní spojení začíná komunikace skriptu a kontrolera. Komunikace probíhá výměnou zpráv, serializovaných pomoci knihovny JSON.

První zprávu, kterou posílá skript, je „Pcinfo“. Kontroler ihned po zjištění, že se napojil nový detekční skript, odešle mu zprávu „Konfigurace“. „Konfigurace“ je první zpráva, kterou přijímá skript. Obsah zprávy určuje režim, ve kterém poběží detekční skript, nastavuje parametry daného režimu.

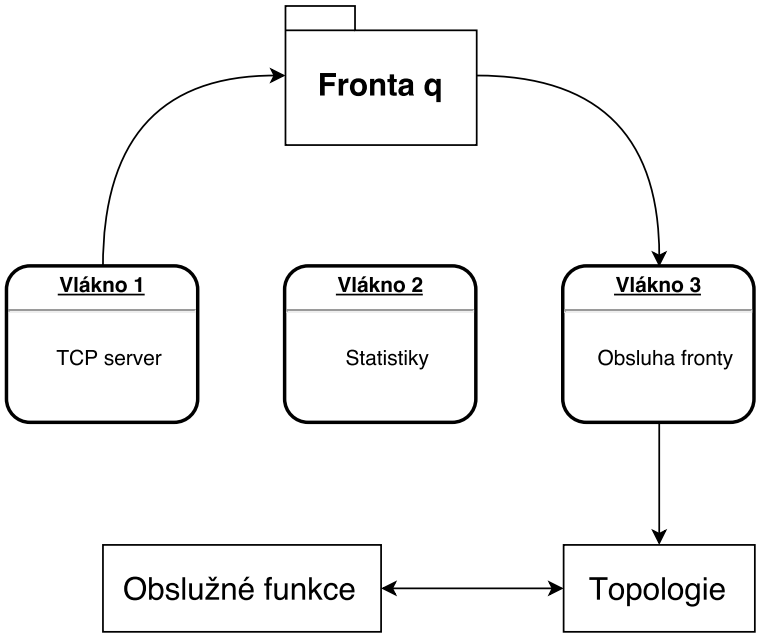
Hlavní logiku „push“ režimu implementuje funkce handle_push_request, která v cyklu v pravidelných časových intervalech zjišťuje aktivní spojení pomocí systémových utilit. Utility se liší podle typu operačního systému a proto se používají jiné na každém z nich. Po získání tabulky aktivních spojeni jednou z nich, se výsledek čistí a parsuje do tvaru tabulky. Následně se daná tabulka filtruje, aby v tabulce nových spojení, jež se pošle na kontroler, byla jen ta, která se považují za prioritní. V dalším kroku cyklu je tabulka starých aktivních spojení obnovena novou. V případě, že se žádné nové spojení neobjevilo, na kontroler se pošle pouze prázdná zpráva, která funguje jako identifikátor pro zjištění, zda je spojení s kontrolerem aktivní, anebo se má skript pokusit napojit na kontroler znovu.

PortStatsReceived	Informování o přeneseném počtu bajtů na vybraném rozhraní.
ConnectionUp	Připojení nového přepínače na kontroler. Součástí je seznam portů na přepínači.
ConnectionDown	Odpojení přepínače od kontroleru. Součástí je seznam portů na přepínači.
PortStatus	Vypnutí rozhraní na přepínači.
LinkEvent	Detekování připojení/odpojení linky mezi dvěma přepínači.
FlowRemoved	Informování o odstranění pravidla z OpenFlow tabulky
HostEvent	Detekování nové IP adresy v síti.

Tabulka 6.1: Seznam zpracováváných a odchytáváných událostí.

6.2 Řídící aplikace pro směrování dat

Hlavním cílem této práce je vytvoření skriptu pro řízení sítě SDN, který implementuje směrování datových toků dle požadavků síťových aplikací. Běh aplikace je rozdělen do tří vláken: TCP server (funkce tcp_server), Statistiky (funkce stats_thread) a Obsluha fronty (funkce handle_queue). Kromě vláken obsahuje aplikace implementaci obslužných funkcí, které jsou volány kontrolerem při výskytu externích událostí. Stav celé topologie je uložen v třídě Topology, která je prostřednictvím zámků zabezpečena pro mezivláknovou komunikaci. Rozdělení aplikace je zobrazeno na obrázku 6.1. Následující text popisuje jednotlivé části aplikace.



Obrázek 6.1: Architektura implementované aplikace.

6.2.1 Implementace severního rozhraní

Aplikace pro řízení sítě implementují severní rozhraní kontroleru, jejichž forma je definovaná obslužnými funkcemi. Obslužné funkce reagují na asynchronní události vyvolané kontrolerem. Následující tabulka popisuje seznam událostí, které výsledná aplikace odchytává a zpracovává:

Zpracování jednotlivých událostí je implementováno následovně:

PortStatsReceived

Na každý přepínač v síti se odesílá dotaz, který zjišťuje sumy odeslaných bajtů pro všechna jeho rozhraní. Na tyto dotazy přepínač odpovídá vygenerováním zprávy. Přijetí zprávy kontroler oznamuje vyvoláním dané události. Využití portů na přepínači se vypočítává podle sumy přenesených bajtů.

ConnectionUp

Ve chvíli, kdy přepínač navazuje spojení s kontrolerem, se vygeneruje událost. Tato událost obsahuje seznam portů na přepínači spolu s jejich parametry. Jedním z parametrů je maximální rychlost linky. Znalost tohoto parametru se vyžaduje pro správnou funkcionalitu směrovacích algoritmů.

ConnectionDown

Po ukončení spojení mezi přepínačem a kontrolerem jsou odstraněny všechny prioritní toky, které protékaly daným přepínačem a také všechny cesty výchozího směrování. Po odstranění starých výchozích cest se vytvoří nové, které budou nahrány na přepínače.

PortStatus

Po vypnutí portu na přepínači se zjistí, jaká IP adresa byla na daném portu připojena. Všechna směrovací pravidla pro danou IP adresu se odstraní ze všech přepínačů v síti.

LinkEvent

Událost vyvolává POX modul openflow.discovery, který se používá na detekci linek mezi přepínači. Daný modul využívá zprávy LLDP, které stále odesílá na všechny porty a následně čeká na jejich příjem. Po detekci nové linky, nebo odpojení stávající, se všechny výchozí cesty vymažou a vytvoří se nové.

FlowRemoved

OpenFlow záznamy pro prioritní aplikace mají na přepínačích nastaven časovač (300 sekund. Pokud v rámci nastaveného časovače nebude přijat žádný paket, který by mohl být zpracován daným pravidlem, bude pravidlo odstraněno. Po jeho smazání bude zároveň odstraněna informace o existenci daného pravidla na kontroleru.

HostEvent

Událost vyvolává POX modul host_tracker, který analýzou ARP paketů vytváří mapování mezi IP adresou, MAC adresou a portem (umístnění) v síti. Po detekci připojení nové IP adresy modul vyvolá událost. Na základě této události se vytvoří výchozí pravidla pro směrování paketů, která budou odeslána na novou IP adresu.

6.2.2 Zjištění vytížení linek

Protokol OpenFlow verze 1.0 neumožňuje přímo zjistit vytížení linek, resp. portů na přepínačích. Informaci o vytížení linek je však nutné získat pro správné fungování směrovacích algoritmů. Pro zjištění vytíženosti se používá atribut, který udává počet odeslaných bajtů na fyzickém portu síťového přepínače. Na kontroleru běží samostatné vlákno, které se každých 5 sekund dotazuje na statistiky všech portů přepínače v síti.

Zpráva s počtem odeslaných bajtů se zpracovává pomocí obslužné funkce PortStatsReceived(). Funkce vypočítá rozdíl počtu přenesených bajtů od posledního nahlášeného údaje. Po vydělení vypočteného rozdílu časem (5 sekund), uběhlého od posledního údaje, se získá průměrná přenosová rychlost.

6.2.3 Integrace detekčních skriptů

Obsluha detekčních skriptů je na kontroleru realizována pomocí samostatného vlákna s názvem tcp_server(). Toto vlákno vytvoří serverový TCP socket, na který se připojují skripty. Po připojení detekčního skriptu na TCP socket a následné výměně počátečních zpráv (jak bylo popsáno v podsekci 4.2.1), se začnou odesílat síťové toky vybraných aplikací na kontroler. Kontroler všechna přijatá data ze socketu ukládá do pomocného řetězce. Vyhledávaní jednotlivých zpráv v daném řetězci probíhá pomocí funkce find_message(). Přijaté zprávy jsou zakódovány ve formátu JSON.

Po dekódování zprávy z formátu JSON se ve zprávě vyhledá identifikátor toku (pětice) spolu s názvem aplikace, které daný tok patří. Získané údaje se uloží do fronty q, která umožňuje bezpečnou komunikaci mezi vlákna.

6.2.4 Spracování nově detekovaných toků

Pro zabránění neustálého nahrávání nových pravidel na síťové přepínače, jsou pravidla pro nově detekované toky vytvářeny po balících a to v pravidelných intervalech. Vlákno handle_queue() každou sekundu kontroluje frontu q, která obsahuje nově detekované pětice s názvem aplikace. Pro každý tok se vyhledají nejlepší cesta mezi odesílatelem a příjemcem síťového toku, pomocí vhodného algoritmu. Podle nalezené cesty se vytvoří a nainstalují příslušná OpenFlow pravidla.

6.2.5 Uložení dat

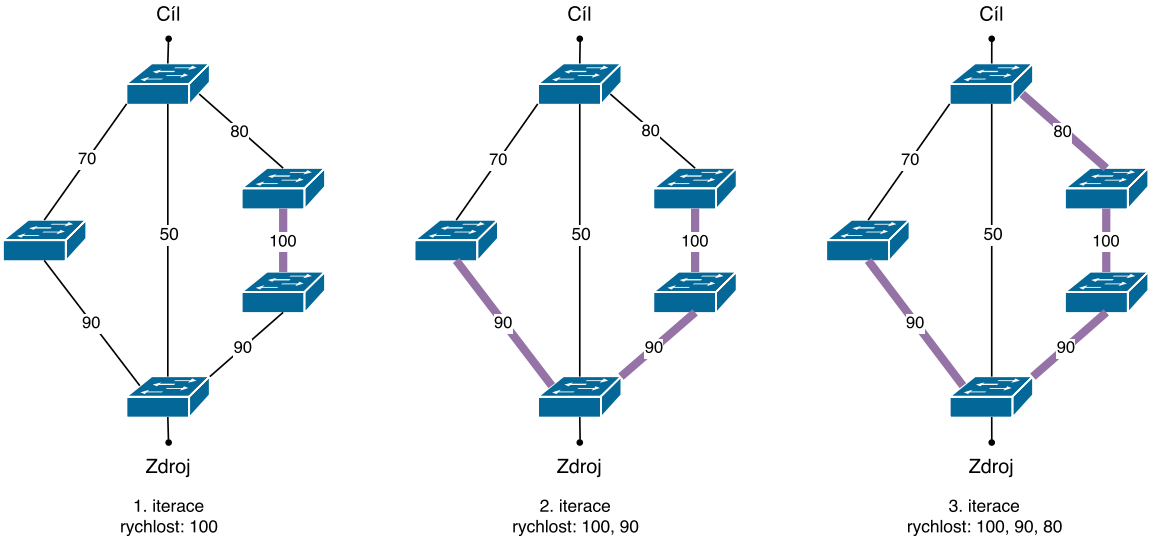
Všechny informace, které skript ke své činnosti potřebuje, jsou uloženy ve slovnících třídy Topology. Jedná se o slovníky:

- **links** - seznam linek mezi přepínači. Obsahuje maximální šířku pásma, aktuální volnou šířku pásma a údaje potřebné k výpočtu dostupné šířce pásma.
- **ip_addresses** - seznam detekovaných IP adres spolu s jejich polohou v síti.
- **switches** - seznam všech připojených přepínačů, objekt spojení k přepínači, seznam portů na přepínači spolu s informací, kam porty vedou a seznam nainstalovaných OpenFlow pravidel.
- **speed** - seznam všech portů v síti a jejich maximální dostupná kapacita.

6.2.6 Směrovací algoritmy

Algoritmus pro nalezení cesty s největší kapacitou (funkce algorithm bandwidth)

Všechny linky v síti se seřadí od největší kapacity po nejmenší podle maximální kapacity nebo aktuální volné kapacity linky. Z linek s největší kapacitou se vytvoří graf, ve kterém se vyhledá nejkratší cesta mezi přepínači, v nichž jsou zapojeny odesílatel a příjemce zprávy. V případě, že cesta není nalezena, se do grafu postupně přidávají linky s nižšími rychlostmi, jak zobrazuje obrázek 6.2.



Obrázek 6.2: Ukázka algoritmu pro nalezení cesty s největší kapacitou.

Algoritmus pro nalezení nejkratší cesty (funkce algorithm delay)

Ze všech linek v síti, bez ohledu na jejich kapacitu, se vytvoří graf, ve kterém se hledá cesta s nejmenším počtem využitých linek.

Algoritmus pro nalezení nejkratší cesty s dostatečnou kapacitou (algorithm both)

Všechny linky v síti se seřadí od největší kapacity po nejmenší podle aktuální volné kapacity linky. Z linek s kapacitou větší, než aplikace vyžaduje, se vytvoří graf. Pokud se v grafu nenajde nejkratší cesta mezi odesílali a příjemcem, začnou se do grafu přidávat linky s menší kapacitou (stejně jak u algoritmu algorithm_bandwidth).

6.2.7 Přeposílaní ARP zpráv

Vytvořené směrovací algoritmy řeší přeposílání paketů mezi zdrojovou a cílovou IP adresou. Pro umožnění jakékoli komunikace mezi adresami musí obě stanice znát cílovou MAC adresu, která se použije při vytváření Ethernetové hlavičky. Tuto adresu stanice získávají protokolem ARP[1]. Pomocí daného protokolu se stanice musí umět domluvit. Použitý POX

[1] https://tools.ietf.org/html/rfc826

modul arp_responder implementuje směrování ARP zpráv prostřednictvím jejich odesílání na všechna síťová rozhraní. Protože v síťové topologii může existovat smyčka, je takové chování bez dodatečného ošetření nežádoucí. Pro odstranění tohoto problému byl vytvořen algoritmus odstranění smyčky v topologii.

Algoritmus pro odstranění smyček (arp_stp()) vytvoří z aktuální topologie pomocí algoritmu STP strom. Algoritmus STP je implementován pomocí funkce minimum_spanning_tree() knihovny networkx. Po vytvoření stromu sítě se vyhledají ty linky topologie, které nejsou ve stromu. Pro tyto linky se vytvoří pravidla, která zahodí všechny příchozí ARP zprávy. Pomocí pravidel vytvořených takovým způsobem, se odstraní všechny možné smyčky protokolu ARP.

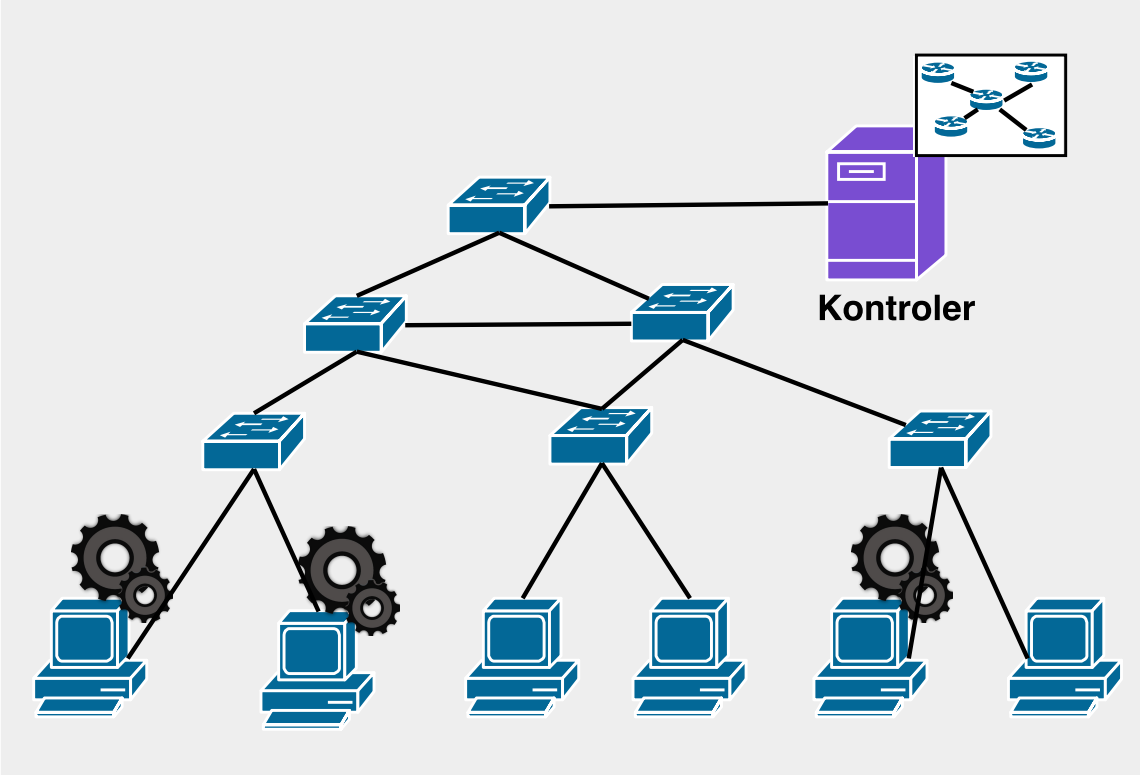
6.2.8 Využití vestavěných POX modulů

Implementovaná řídící aplikace pro správný chod vyžaduje spuštění více vestavěných POX modulů. Aplikace s ostatními moduly přímo nekomunikuje, ale zpracovává události, které moduly generují. Popis použitých vestavěných modulů je následující:

- **openflow.discovery** - Detekce linek mezi přepínači. Pomocí znalostí přepínačů a linek mezi přepínači je skript schopen vytvořit topologii sítě.
- **host_tracker** - Detekce zapojených IP adres v síti pomocí které se do topologie sítě přidají koncová zařízení.
- **proto.arp_responder** - Modul obsluhující ARP zprávy, které přeposílá v síti. Pomocí tohoto modulu jsou koncové stanice schopny komunikovat protokolem ARP.

Vestavěný modul host_tracker z neznámých příčin občas neohlásil IP adresu pro nově detekovány stanice, ale pouze jejich polohu a MAC adresu. Modul byl upraven přidáním několika řádků kódu.

Architektura výsledného řešeni zobrazující detekční skripty na koncových stanicích a aplikace pro směrování běžící na kontroleru je zobrazena na obrázku 6.3.

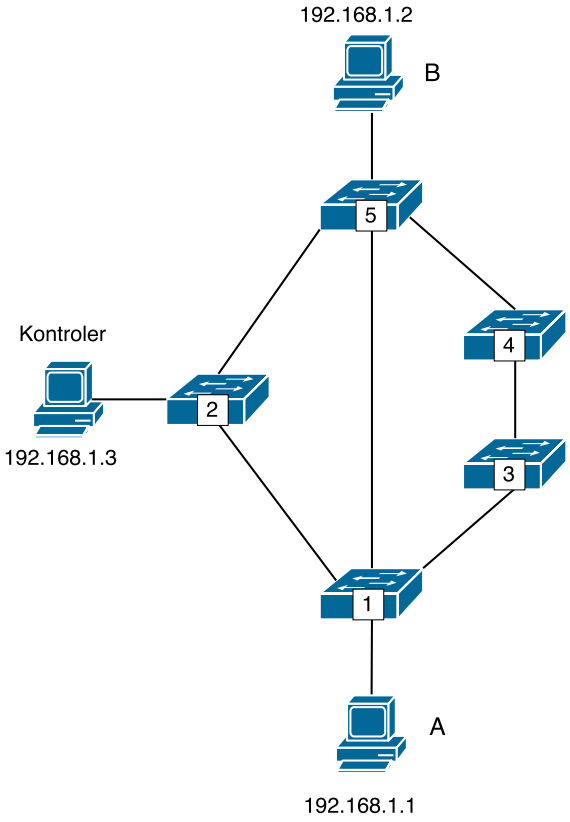


Obrázek 6.3: Architektura výsledného řešeni.

Kapitola 7

Testování

Testování implementovaných aplikací probíhalo ve virtualizovaném prostředí Mininet. Mininet umožňuje vytvořit libovolnou topologii obsahující OpenFlow přepínače. Do vytvořené topologie byly zapojeny virtuální počítače, na kterých běžel operační systém Windows 10 a Ubuntu 16.04. Na obou počítačích byl nainstalován detekční skript pro ohlašování nových síťových toků. Součástí topologie byl kontroler POX, který kromě řídícího spojení s přepínači musel mít také spojení s koncovými stanicemi. Testovací topologie je zobrazena na obrázku 7.1.



Obrázek 7.1: Testovací topologie.

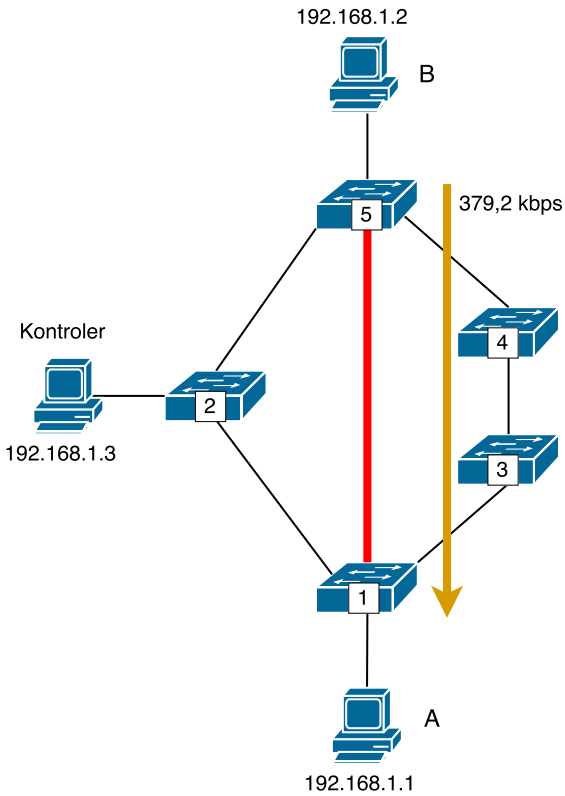
Po vytvoření topologie byl spuštěn kontroler POX, na který se postupně začaly připo-

jovat vytvořené přepínače. Kontroler POX byl spustěn s parametry:
sudo python pox.py log.level –WARNING openflow.discovery penflow.topology host_tracker samples.bp proto.arp_responder
Po připojení všech přepínačů ke kontroleru se začala spouštět koncová zařízení. Všechna koncová zařízení měla nastavenou statickou IP adresu podle obrázku 7.2. Detekční skripty na počítačích A a B byly nakonfigurovány, aby se připojovaly k IP adrese 192.168.1.3.

Pro jednodušší testování byla rychlost všech linek mezi přepínači omezena na 10Mbps. Linky ke koncovým zařízením zůstaly ve výchozím nastavení poskytujícím rychlost 10Gbps. Před zahájením testů měřících rychlost přenosu dat byla otestována vzájemná konektivita koncových stanic pomocí nástroje Ping. Vzhledem k úspěšnému testu se mohlo přejít k hlavním testům.

7.1 Test 1

Cílem testu je otestovat výběr použité linky pro aplikaci nevyžadující prioritní zpracování. Během testu je linka mezi přepínačem 1 a 5 zatížena přenosem rychlostí 8Mbps. I při nastavení maximální rychlosti linek 10Mbps nebylo možné dosáhnout větší rychlost jako 8Mbps. Obrázek 7.2 zobrazuje směr toku dat s přenosovou rychlostí.



Obrázek 7.2: Schéma prvního testu.

Popis testu:

1. Z počítače A se spustilo kopírování velkého souboru z počítače B ($B \rightarrow A$), čímž vzniklo vytížení linky mezi přepínači 1 a 5 počas celé doby testu.Linka 1 – 5 byla vy-

tížena, protože výchozí směrovací pravidla pro přenos dat mezi IP adresami 192.168.1.1 a 192.168.1.2 směrují přenos cestou 192.168.1.1 – 1 – 5 – 192.168.1.2.testu:

2. Na počítači A se spustí aplikace FileZilla, pomocí které je zahájen přenos dalšího souboru z počítače B ve směru $B \rightarrow A$.

3. Rychlost přenosu souboru se pohybuje kolem 379,2kbps.

4. Analyzováním OpenFlow tabulek bylo zjištěno, že přenos druhého souboru aplikací FileZilla využívá již vytíženou linku 1 – 5.

7.2 Test 2

Jedná se o zopakování testu č. 1, avšak s využitím detekčního skriptu pro nalezení prioritního síťového toku. Nastavením požadavku pro maximální rychlost přenosu aplikaci FileZilla se očekává zrychlení přenosu souboru z počítače B počítači A.

1. Z počítače A se spustilo kopírování velkého souboru z počítače B ($B \rightarrow A$), čímž vzniklo vytížení linky mezi přepínači 1 a 5 během celé doby testu. Linka 1 – 5 byla vytížena, protože výchozí směrovací pravidla pro přenos dat mezi IP adresami 192.168.1.1 a 192.168.1.2 směrují přenos cestou 192.168.1.2 > 1 - 5 > 192.168.1.1.

2. Na počítači A se spustí aplikace FileZilla, pomocí které je zahájen přenos dalšího souboru z počítače B ve směru $B \rightarrow A$.

3. Rychlost přenosu souboru se znovu pohybuje kolem 354,8kbps.

4. Po pár vteřinách se začne rychlost přenosu zvyšovat, až dosáhne hodnotu 8104kbps.

5. Přenos souboru probíhá cestou 192.168.1.2 > 2-5 > 1-2 > 192.168.1.1, která sice není nejkratší, ale poskytuje nejvyšší možnou přenosovou rychlost.

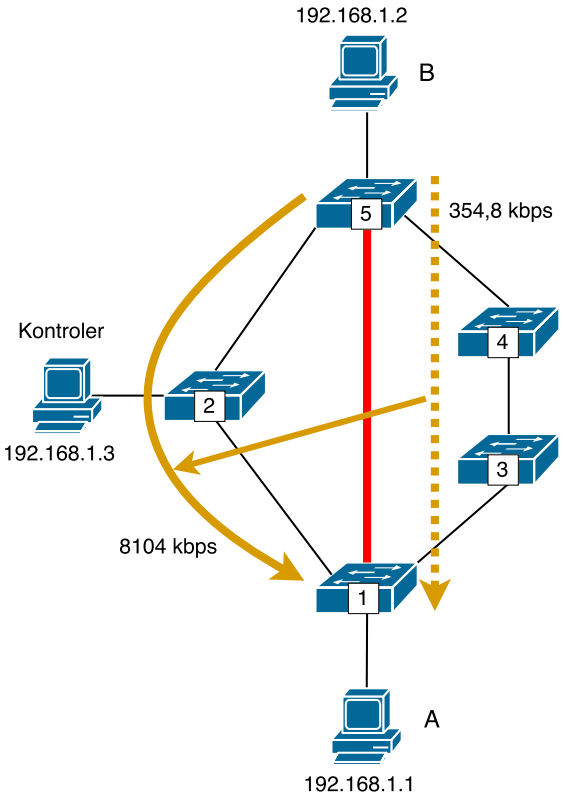
Důvodem nízké rychlosti v kroku číslo 3 je, že detekce prioritního síťového toku a následné vytvoření OpenFlow pravidel trvá až několik vteřin. Na obrázku 7.3 je tato situace zobrazena prostřednictvím dvou cest přenosu dat. Před detekováním aplikace se využívá cesta znázorněná přerušovanou čarou poskytující rychlost přenosu 354,8kbps. Po nalezení optimálnější cesty se použije cesta znázorněná plnou čarou. Po přepnutí směrovací cesty optimálními linkami se přenosová rychlost zvýšila na 8104kbps.

7.3 Test 3

Při tomto testu je kromě linky 1-5 vytížená linka mezi přepínači 2 a 5. Linka 2-5 je vytížena rychlostí 5Mbps, takže je možné přes linku přenést další 3Mbps. Požadavky aplikace FileZilla byli změny na nejkratší cestu s minimální rychlostí 2Mbps.

1. Linka 1-5 je maximálně vytížená s volnou kapacitou 0kbps. Linka 2-5 je částečně vytížená, umožňující přenos dat rychlostí přibližně 3Mbps. Všechny ostatní linky jsou nevytížené, umožňující přenos dat rychlostí 8Mbps.

2. Na počítači A se spustí aplikace FileZilla, pomocí které je zahájen přenos dalšího souboru z počítače B ve směru $B \rightarrow A$.



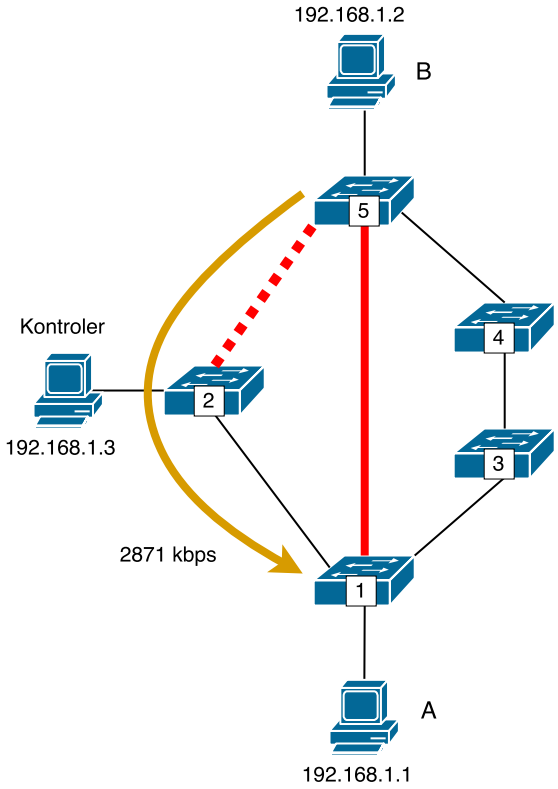
Obrázek 7.3: Schéma druhýho testu.

3. Rychlost přenosu se po krátkém čase ustálí na hodnotě 2871kbps.

4. Analýzou OpenFlow tabulek byla nalezena použitá cesta pro přenos dat. Jednalo se o cestu 192.168.1.2 > 1 - 5 > 192.168.1.1.

Obrázek 7.4 zobrazuje výslednou zvolenou cestu, která není nejkratší a ani nejrychlejší cestou. Jedná se o kombinaci obou parametrů. Byla použita nejkratší možná cesta s volnou kapacitou alespoň 2Mbps pro přenos.

Cílem testů bylo proukázat výhody směrování dle typu přenášené aplikace. Při neznalosti typu aplikace, byla zvolena směrovací cesta, která ne vždy odpovídala nejoptimálnější variantě. Když byla aplikace FileZilla nastavena, jako aplikace vyžadující maximální rychlost přenosu dat, rychlost přenosu se výrazně zvýšila. Na základě provedených testů lze výsledné aplikace označit jako úspěšné.



Obrázek 7.4: Schéma třetího testu.

Kapitola 8

Závěr

V rámci své bakalářské práce jsem se seznámila s principem softwarově definovaných sítí a možnostmi, které nabízejí pro zefektivnění a zlepšení směrování dat v síti. Cílem mojí práce bylo navrhnout a naimplementovat směrovací algoritmus pro SDN kontroler a také detekční mechanizmus pro koncové stanice. Pro implementaci směrovací aplikace byl zvolen kontroler POX pro svou jednoduchost v rozšíření a testování nových funkcí. Obě aplikace byly implementovány v jazyce Python.

Výše zmíněné aplikace byly funkcionálně a logicky propojené, naimplementované a následně otestované v emulačním nástroji Mininet.

Výsledkem práce jsou dvě aplikace. Pro směrovací aplikaci na kontroler byly navrhnuty a naimplementovány celkem čtyři směrovací algoritmy. Jeden algoritmus jako defaultní pro směrování všech neznámých toků a také toků neprioritních aplikací. Seznam prioritních aplikací byl definován síťovým administrátorem na kontroleru a následně odeslán jako jeden z parametrů konfigurační zprávy pro detekční skript. Ostatní tři algoritmy berou své základy z defaultního algoritmusu a jsou určené pro směrování dat prioritních aplikací nejlepší možnou cestou s ohledem na požadavky aplikací na síť (šířka pásma, zpoždění).

Detekční skript byl navrhnut a naimplementován jako démon skript, který může běžet na pozadí koncové stanice a sbírat informace o síťových tocích. Pro získávání dané informace jsou nová spojení odfiltrovaná na jen prioritní a odeslané na kontroler.

Chování aplikací, které je popsané výše, bylo demonstrováno experimenty, jejichž výsledky dokazují, že aplikace fungují správným způsobem. Pro aplikace je možné navrhnout rozšíření ve formě dalších režimů detekčního skriptu. Jedním z takých rozšíření může byt režim „pull“. Princip jeho fungování lze popsat takto: v případě, že na kontroler bude přeposlán paket z neznámého toku dat, kontroler se optá detekčních skriptů, zda daný datový tok (ve formě pětice) nepoznají. V případě kladné odpovědi detekční skript vyhledá jméno aplikace, která spravuje daný tok, připojí ho k pětici a odešle zpátky na kontroler. Výhodou daného rozšíření by bylo menší zatížení kontroleru při směrování neznámých paketů.

Literatura

[1] Betts, M.; DAvis, N.; Dolin, R: Software-Defined Networking (SDN) Definition. URL <http://goo.gl/O2eTti>

[2] Chao, H. J.; Liu, B.: *High performance switches and routers*. John Wiley & Sons, 2007.

[3] Comer, D. E.: *Computer Networks and Internets with Internet Applications*. Prentice Hall, páté vydání, 2009, ISBN 0-13-606698-4.

[4] Farhady, H.; Lee, H.; Nakao, A.: Software-defined networking: A survey. *Computer Networks*, ročník 81, 2015: s. 79–95.

[5] Feamster, N.: Motivation for "Northbound APIs" and SDN Programming Languages? online, module 6.1, [cit. 2016-05-1]. URL <https://class.coursera.org/sdn-001/lecture/69>

[6] Huitema, C.: *Routing in the Internet*. Prentice Hall PTR, 1999.

[7] Jacquet, P.: Optimized link state routing protocol (olsr). 2003.

[8] Johnson, S.: A primer on northbound APIs: Their role in a software-defined network. *SearchSDN*, December 2012. URL <http://searchsdn.techtarget.com/feature/A-primer-on-northbound-APIs-Their-role-in-a-software-defined-network>

[9] Marschke, D., Doyle J., Moyer P.: What is SDN, booktitle =.

[10] Mccauley, J.: Pox: A python-based openflow controller. 2014.

[11] Nadeau, T. D.; Gray, K.: *SDN: software defined networks*. "O'Reilly Media, Inc.", 2013.

[12] Pusateri, T.; aj.: Distance vector multicast routing protocol. *Work in Progress. Authors' Addresses J. Mark Pullen C3I Center/Computer Science Mail Stop 4A5 George Mason University Fairfax, VA*, ročník 22032, 2003.

[13] Rabaey, J. M.; Potkonjak, M.; Koushanfar, F.; aj.: Challenges and opportunities in broadband and wireless communication designs. In *Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on*, IEEE, 2000, s. 76–82.

[14] Rouse, M.: Definition northbound interface / southbound interface. November 2012. URL <http://whatis.techtarget.com/definition/northbound-interface-southbound-interface>

[15] SDxCentral: What are SDN Southbound APIs? [online], 2012-2016 [cit. 2016-05-1].
URL
<https://www.sdxcentral.com/resources/sdn/southbound-interface-api/>

[16] SDxCentral: What is OpenFlow? Definition and how it relates to SDN? [online], 2012-2016 [cit. 2016-05-2].
URL <https://www.sdxcentral.com/resources/sdn/what-is-openflow/>

[17] Shin, S.; Porras, P. A.; Yegneswaran, V.; aj.: FRESCO: Modular Composable Security Services for Software-Defined Networks. In *NDSS*, 2013.

[18] Yu, M.; Jose, L.; Miao, R.: Software Defined Traffic Measurement with OpenSketch. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, s. 29–42.