VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Implementace Cisco One Platform Kit (onePK) v počítačových sítích s QoS

Implementation of Cisco One Platform Kit (onePK) in computer networks

2017 Petr Antončík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Petr Antončík**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T059 Mobilní technologie

Téma: Implementace Cisco One Platform Kit (onePK) v počítačových sítích s QoS
Implementation of Cisco One Platform Kit (onePK) in Computer Networks

Jazyk vypracování: čeština

Zásady pro vypracování:

Přenos dat počítačovou sítí vykazuje různé vlastnosti a pro určité typy přenosů je nutno zajistit definované parametry. V souhrnu se jedná o kvalitu služeb QoS. Nastupujícím trendem jsou softwarově definované sítě. Námětem této práce je modifikace parametrů směrovací sítě pro zajištění určité kvality služby QoS. Práce bude obsahovat:

1. Popis nástroj Cisco One Platform Kit (onePK) a jeho možnosti.
2. Popis QoS.
3. Návrh různých algoritmů, které zajistí kvalitu služeb, QoS.
4. Simulace navržených algoritmů QoS v modelu směrovací sítě.
4. Vyhodnocení simulace.

Seznam doporučené odborné literatury:

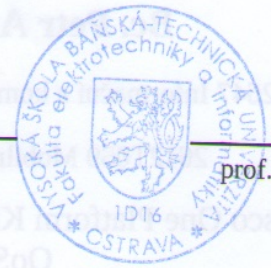
[1] Tim Szigeti, Christina Hattingh, Robert Barton, Kenneth Briley Jr.: End-to-End QoS Network Design: Quality of Service for Rich-Media & Cloud Networks, 2 edition; Cisco Press 2013; ISBN 978-1587143694
[2] Paul Goransson, Chuck Black: Software Defined Networks: A Comprehensive Approach 1st Edition; Morgan Kaufmann, 2014; ISBN 978-0124166752

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Ing. Jaroslav Zdrálek, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017



doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 ..... Petr Antoušek .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 28. dubna 2017 ..... Petra Antošová

Rád bych na tomto místě poděkoval panu doc. Ing. Jaroslavu Zdrálkovi, Ph.D., který mi dopomohl ke vzniku této práce. Dále bych rád poděkoval své rodinně a spolupracovníkům, kteří mi byli oporou při sepisování této práce.

Abstrakt

Tato práce obsahuje postup pro vytvoření topologie využívající onePK rozhraní pro správu a automatizaci sítě. Součástí práce je ukázka tří algoritmů, které využívají tato programovatelná rozhraní a pomocí nich zajišťují kvalitu služeb v dané síti. Práce obsahuje informace o tom, co vše je třeba nastavit v rámci virtuálního prostředí, a jak je potřeba postupovat, aby nám komunikace mezi algoritmy a jednotlivými směrovači fungovala.

Klíčová slova: onePK, QoS, Java, Eclipse, API, síť, směrovač, konfigurace, IOS, topologie

Abstract

This work include principes for creating topology which working with onePK APIs to ensure management and automation in network. This work also include three algorithms which use these programmable APIs for quality of services in this network. Part of the work is information what is necessary to setup in this virtual environment and step by step to create communication between algorithms and routers.

Key Words: onePK, QoS, Java, Eclipse, API, network, router, configuration, IOS, topology

Obsah

Seznam použitých zkratek a symbolů **10**

Seznam obrázků **11**

Úvod **12**

1 onePK **14**
1.1 Architektura onePK 14
1.2 Vývoj 14
1.3 Balíčky služeb 15
1.4 Linux container 16
1.5 LISP 16

2 QoS **17**
2.1 Mechanismy QoS 17
2.2 Parametry řešené v rámci QoS 18
2.3 Akce prováděné pro zajištění kvality služby 20

3 Prostředí onePK **23**

4 Virtuální topologie **25**
4.1 Vytvoření skriptů pro spuštění 25
4.2 Vytvoření topologie 28
4.3 Spuštění topologie 31

5 Přesměrování provozu během výskytu CRC chyb na lince **35**
5.1 Naprogramované funkce 35
5.2 Testování programu 38
5.3 Chování topologie při použití programu 40

6 Změna parametrů QOS v čase **42**
6.1 Naprogramované funkce 42
6.2 Testování programu 45
6.3 Chování topologie při použití programu 46

7 Přenastavení hodnot traffic shapingu podle vytížení linky **49**
7.1 Naprogramované funkce 49
7.2 Testování programu 50
7.3 Chování topologie při použití programu 52

8 Závěr **54**

Literatura **56**

9 Obsah CD **58**

Přílohy **58**

A Elektronické přílohy **59**
A.1 Zdrojové skripty pro sestavení topologie 59
A.2 Zdrojové skripty s návrhem topologie 59
A.3 Zdrojové kódy programů 59
A.4 Obrázky pořízené během testování 59

B Přílohy **60**
B.1 xnode.py 60
B.2 xnode_certify.sh 60
B.3 xnode_create.sh 62
B.4 xnode_delete 62
B.5 xnode_info.sh 63
B.6 netlist.sh 63
B.7 xnode.virl 63
B.8 Konfigurace 67

Seznam použitých zkratek a symbolů

QoS	–	Quality of Services
onePK	–	Open network Environment Platform Kit
API	–	Application Programming Interface
DSCP	–	Differentiated Services Code Point
TCP	–	Transmission Control Protocol
TSL	–	Transport Layer Security
CoS	–	Class of Service
ACL	–	Access Control List
RIB	–	Routing Information Base
VTY	–	Virtual Teminal Lines
LISP	–	Cisco Locator ID Separation Protocol
RSVP	–	Resource Reservation Protocol
RED	–	Random Early Detection
WTD	–	Weighted Tail Drop

Seznam obrázků

1.1 Možnosti umístění aplikace. 15
4.1 Fyzické propojení prvků v síti. 31
4.2 Vytvoření topologie přes konzolový příkaz. 32
4.3 Ukončení topologie přes konzolový příkaz. 32
4.4 Dialogové okno průběhu spouštění. 33
4.5 Zavádění systému do směrovače č.2. 33
4.6 Vytížení systému během zavádění systému IOS. 34
5.1 Logické zapojení topologie při testování. 39
5.2 Výstup ze směrovače pro linku bez fyzických chyb. 40
5.3 Výstup ze směrovače pro linku při výskytu chyb. 41
6.1 Logické zapojení topologie při testování QOS v závislosti na čase. 46
6.2 Aplikování QOS profilu 1 na rozhraní. 47
6.3 Aplikování QOS profilu 2 na rozhraní. 48
7.1 Logické zapojení topologie při testování programu pro traffic shaping. 51
7.2 Aplikování profilu 1 pro traffic shaping na první rozhraní. 52
7.3 Aplikování profilu 2 pro traffic shaping na první rozhraní. 53

Úvod

V rámci počítačových sítí a síťových zařízení existuje mnoho výrobců těchto zařízení, a také mnoho standardů, které se používají pro přenos informací v rámci síťových topologií. Není divu, že se každý výrobce těchto zařízení snaží na trhu prosadit a nabízí to své jako nejlepší a nejspolehlivější. Jelikož je výrobců těchto zařízení poměrně mnoho, tak vznikl trend zařízení vylepšovat nejen po stránce hardwarového výkonu, ale taky po stránce softwaru. Z oblasti hardwaru došlo k rozdělení zařízení na tři typy fixed, stackable a modulární. Fixed zařízení jsou ta nejběžnější, která můžeme na trhu vidět. Jedná se o zařízení s pevně danou konfigurací portů a hardwarových prvků, které nelze nijak rozšiřovat. Proti tomu se staví tak zvaná modulární zařízení. To jsou ta, kde můžeme vyměňovat různé moduly, které obsahují jiné porty a jiné hardwarové komponenty, a tím nám umožňují vylepšovat síťové zařízení a zvyšovat výkon v rámci hardwaru. Nevýhodou je, že jsou o poznání dražší než běžné fixní. Mezi těmito dvěmi pak stojí speciální skupina nazývaná stackable. Jedná se vlastně o fixní zařízení se speciálními porty, které umožňují v případě nutnosti propojit tato zařízení mezi sebou a zvýšit tak výkon a možnosti. Takto propojená zařízení se chovají jako jeden celistvý prvek. V rámci softwaru byly do zařízení implementovány různé standardy a podporované protokoly, které by tyto zařízení měly podporovat a zajistit tak kompatibilitu s co nejširším segmentem zařízení. Kromě těchto standardů a protokolů, které nám tyto zařízení poskytují, se zde také objevuje otázka konfigurace a zprovoznění těchto zařízení. Tato zařízení mohou administrátoři konfigurovat nejen lokálně přes konzoli, ale taky pomocí vzdáleného přístupu přes již vytvořenou síť pomocí telnetu nebo ssh přístupu. V rámci usnadnění konfigurace a její automatizace se firma Cisco rozhodla vytvořit nové programovatelné API rozhraní nazývané onePK umožňující programátorům vytvářet programy, které komunikují přímo se zařízeními této firmy. Takto fungující síť nám pak přináší mnoho výhod, protože správu sítě nám mohou vytvářet naprogramované programy, a zároveň je také kladen menší nárok na samotného správce sítě.

Ve své diplomové práci se zaměřím především na to, jak vytvořit síť z těchto Cisco zařízení podporujících onePK a vytvořím program, který bude provádět automatickou správu sítě přes tato API rozhraní. Zároveň se budu snažit navrhnout a vytvořit tento program pro tuto síť tak, že kromě bežné správy bude také síť monitorovat a na základě zjištěného provozu a sezbíraných dat bude řešit Quality of Services v této topologii. Popíšu zde, jaké prostředky budu potřebovat, abych tuto síť vytvořil, co vše a jaké služby budu muset nakonfigurovat a spustit pro vytvoření sítě využívající plně onePK rozhraní. V první části diplomové práce rozeberu teorii, jak tato onePK API fungují v rámci Cisco zařízení, a jak skrze ně správa probíhá. Také uvedu možná řešení QoS v této topologii a na základě jakých parametů je možné QoS ovlivnit. Další část mé práce bude obsahovat praktický postup, jak vytvořit takovou síť a samotné naprogramování programů, které budou skrz tato API komunikovat se sítí a řešit správu. V poslední části se

zaměřím na praktické vyzkoušení programu na reálně vytvořené topologii spolu s otestováním, jak programy spravují síťová zařízení pro různé situace.

1 onePK

Cisco onePK je zkratka pro Open Network Environment Platform Kit. Jde tedy o síťové rozhraní, které umožňuje vývojářům vyvíjet aplikace snadno integrovatelné s prostředím síťových zařízení vyvinutých firmou Cisco. Jedná se o programovatelný framework, který umožňuje uživatelům upravovat a zasílat data ze sítě mimo tuto síť. Toto rozhraní bylo vyvinuto především za účelem získání přístupu k informacím uvnitř sítě a k umožnění přímé kontroly toků dat a cest uvnitř této sítě. Rozhraní umožňuje uživatelům rychle upravovat a nastavovat danou síť pro kladené požadavky, ať již pro různé aplikace nebo pracovní cíle. V zásadě toto rozhraní obsahuje různá API (aplikačně programovatelná rozhraní) a knihovny, které umožňují programátorům vytvářet vlastní aplikace a které spolupracují se zařízeními od Cisca.[1, 2]

OnePK má za cíl rozšířit možnosti síťových zařízení a sítě, jako je například užší integrace s aplikacemi, lepší ovládání a přípravy síťové infrastruktury pro rychlé vytváření služeb, zvýšení bezpečnosti datových center a poskytovatelů služeb. Dalším cílem je také automatizace v oblasti konfigurací zařízení a probíhajících aktivit.

1.1 Architektura onePK

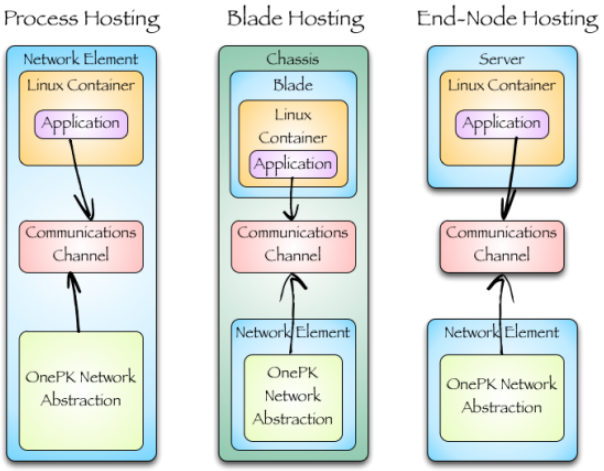
Samotná architektura onePK se skládá ze tří částí. Z prezenční vrstvy, API infrastruktury a komunikačního kanálu. V prezenční vrstvě se nacházejí API knihovny použitelné v rámci tvořených aplikací. API infrastrukrura nám umožňuje přístup k vnitřím funkcím Cisco zařízení, tedy směrovačů a přepínačů od Cisca. Komunikační kanál nám pak poskytuje bezpečný a rychlý způsob přenosu dat mezi aplikaci a síťovým prvkem.[1, 2]

OnePK umožňuje hlubší přístup ke zdrojům a možnostem směrovačů a přepínačů od Cisca pomocí těchto navržených programů. Díky podpoře onePK na všech zařízeních Cisco má programátor možnost vytvořit nějaký program jen jednou a tento program je schopen pracovat s kterýmkoli zařízením v síti. Samozřejmostí je i podpora několika programovacích jazyků pro vytváření aplikací.

1.2 Vývoj

V rámci vytvoření aplikací Cisco rozlišuje tři vlny vývoje, Process hosting, Blade hosting a End-node hosting. U Process hostingu je aplikace vložena v linuxovém prostředí síťového prvku, kde izolaci onePK aplikace a síťových procesů vytváří takzvaný linux konteiner, o kterém se dozvíme později. Druhý způsob jak implementovat naši aplikaci je blade hosting. V tomto případě je aplikace uložena na vyhrazeném hardwaru (blade), který se síťovým prvkem sdílí společné šasi. Poslední možný způsob umístění aplikace je typický síťový neboli End-node hosting. V tomto

případě je aplikace uložena na nějakém koncovém zařízení, nejčastěji serveru odkud aplikace monitoruje a řídí celou síť.[1, 2]



Obrázek 1.1: Možnosti umístění aplikace.
Zdroj: https://blogs.cisco.com/security/ciscos-onepk-part-1-introductio

1.3 Balíčky služeb

V rámci funkcionality Cisco poskytuje dva typy balíčků. Prvním typem jsou základní balíčky služeb, které poskytují základní stupeň funkcionality pro onePK aplikace a druhým typem jsou volitelné, které přidávají navíc k základní funkčnosti nějaké ty rozšiřující funkce, které si zvolíme.

1.3.1 Základní baličky

1. **DataPath Service Set**: Poskytuje aplikacím možnost zachytit pakety procházející směrovači nebo přepínači a kopírovat tyto pakety pro aplikace nebo je pozastavit. Při pozastavení nejsou pakety poslány k cíli, dokud nejsou aplikaci zpracovány.

2. **Policy Service Set**: Poskytuje rozhraní pro nastavení ACL, QoS politik a možnost pozastavení nebo kopírování provozu do Data Path Service setu.

3. **Routing Service Set**: Poskytuje aplikacím rozhraní pro přístup k směrovacím informacím uloženým v síťovém zařízení v jeho RIB tabulce (Routing information base). Samozřejmostí je možnost modifikace těchto dat.

4. **Element Service Set**: Poskytuje model jádra zařízení pro systém a možnost obsluhy aplikace připojené k síťovému zařízení.

5. **Discovery Service Set**: Poskytuje mechnismus pro aplikace k odhalení síťové topologie a zařízení poskytujících onePK služby v síti.
6. **Configuration Managment Service Set**: Poskytuje možnost upravovat konfiguraci a vytvářet změny v již spuštěné konfiguraci.
7. **Event Service Set**: Poskytuje zpětná volání v infrastruktuře pro detekování různých systémových změn a událostí během jejich výskytu.
8. **Developer Service Set**: Poskytuje možnost debugu, vytváření log souborů a prozkoumání histore změn.

1.3.2 Volitelné balíčky

1. **Pathtrace Service Set**: Poskytuje službu Mediatrace. Jedná se o rozšířenou verzi Traceroute nástroje, kdy tento nástroj umí zjistit cestu nejen na třetí vrstvě, jako jsou směrovače, ale také skrz přepínače fungující na druhé vrstvě OSI modelu.
2. **LISP Service Set**: Poskytuje rozhraní pro LISP.
3. **VTY Service Set**: Poskytují službu překrývající služby virtuálních terminálů. Umožňují zasílání příkazu z aplikací, které se mají vykonat na síťovém zařízení skrz VTY rozhraní.[2]

1.4 Linux container

Jedná se o možnost virtualizace rozhraní operačního systému pro spuštění několika izolovaných linuxových systému ovládaných jedním hostem. Linuxový kernel nám poskytuje kontrolní skupiny, které umožňují omezit a prioritizovat hardwarové zdroje bez potřeby spouštět nové virtuální stroje.[2]

1.5 LISP

Jedná se o směrovací architerkturu, která umožňuje vytvářet směrování pomocí znaků. Běžná architektura pro směrování využívá IP adresy pro definování identity zařízení a způsobu, jakým je zařízení připojeno v síti pomocí jediného číselného prostoru. LISP architektura odděluje identitu zařízení (EID) od jeho umístění (RLOC) do dvou číselných prostorů. To nám umožňuje zjednodušení směrování pro multihomed směrování, usnadňuje rozšíření any-to-any WAN připojení a umožňuje mobilitu pro virtuální stroje v datacentrech. Také je docíleno zlepšení rozšiřitelnosti při sjednocení RLOC, optimalizace směrování a snížení složitosti operací během směrování.[2, 3]

2 QoS

QoS neboli Quality of Service je termín využívaný v informatice k popisu řízení, rezervací a správě datových toků vyskytujících se v počítačových sítích založených na paketovém provozu. Díky QoS můžeme manipulovat s přenosovou kapacitou jednotlivých linek, a tak prioritizovat a nastavit množství přenášených dat pro jednotlivé služby, a tím zajistit jejich spolehlivost v případě vyčerpání zdrojů v síti. V rámci této správy můžeme určitým službám, které jsou pro nás důležité, například VoIP, vyhradit určitou přenosovou rychlost, abychom zajistili spolehlivost a neztrácela se nám data. Naopak služby, které nejsou citlivé na ztráty dat, můžeme omezit, aby nedošlo k vyčerpaní zdrojů v síti. Nejčastěji se QoS implementuje ve WAN síti než v LAN. Důvodem je, že v rámci LAN sítí se nachází dostatek zdrojů a zařízení zde nejsou nijak výrazně přetěžována. Naopak WAN sítě spojují velké množství jednotlivých LAN sítí, takže zde dochází ke křížení mnoha datových toků a velkého množství dat. Proto jsou zde vysoké nároky na jednotlivá síťová zařízení, která mohou dosáhnout hranic svých kapacit. V takových případech pak dochází k zahazování některých paketů. Proto je výhodné zde nastavit QoS, aby nebyly zahazovány ty pakety, které nesou důležité informace.[4]

2.1 Mechanismy QoS

V rámci QoS se dnes používají v zásadě tři typy metod zajišťující kvalitu provozu. První metodou je tak zvaná **Best-effort services**. Jedná se o defaultní metodu na všech zařízeních která, jak již vyplývá z názvu, se snaží přijaté pakety co nejrychleji zaslat do cíle a neřeší, o jaký typ paketu se jedná. Jednoduše, který paket přijde do síťového zařízení dříve, ten je také dříve odeslán do cíle.

Druhá metoda se nazývá **Differentiated services** neboli Soft QoS. Z názvu si můžeme všimnout, že se jedná o metodu, která rozděluje jednotlivé pakety podle dané služby do jednotlivých kategorií, kterým QoS přiřazuje jednotlivé vlastnosti. Samotné rozdělení je založeno na hodnotě **DSCP**, což je šestibitové pole v hlavičce IP packetu. Toto pole nám rozhoduje do které skupiny data v paketu patří, a jakým způsobem má být paket zpracován. Například můžeme vytvořit skupinu, kde mají být data rychle zpracovány, protože jsou citlivá na zpoždění jako je například multimédiální tok a hlasový tok, a naopak zase webové služby můžeme nechat v standardu best effort.

Třetí metoda je **Integrated services** neboli Hard QoS. Tato metoda využívá principu rezervace zdrojů v síti a následného posílání dat. Nejprve se rezervují zdroje pro celou cestu od zdroje k cíli pomocí protokolu **RSVP - Resource Reservation Protocol** a až je vyjednán celý přenosový kanál, tak poté jsou data po této cestě zaslána do cíle. Nevýhodou této metody je, že samotné vyjednání rezervace zdrojů trvá nějaký čas a je náročné na zdroje jednotlivých síťových

prvků. Nezbytnou podmínkou je také, že tuto metodu musejí podporovat všechna zařízení na trase přes které je daný rezervovaný kanál zřízen, a to včetně koncových aplikací. Proto se dnes tato metoda již moc nepoužívá.

Z těchto metod, které jsme si popsali se dnes primárně používá metoda **Differentiated services**, jelikož se jedná o metodu, která pouze přeznačuje jednotlivé pakety a samotné řešení jak s pakety zacházet si řeší každý síťový prvek sám. Zároveň značení probíhá rychle a není nutné, aby toto značení uměl vytvářet každý prvek na trase. Taktéž data jsou posílána okamžitě a nemusí čekat na vyjednání kanálu.[5, 6, 7]

2.2 Parametry řešené v rámci QoS

V rámci QoS se řeší pět hlavních parametrů, které mohou mít negativní dopad při přenašení dat pro daný provoz. Jedná se o **přenosovou rychlost**, **zpoždění**, **variabilita zpoždění**, **ztrátovost paketů** a **doručení mimo pořadí**. Tyto parametry nám ovlivňují, jak se jednotlivé datové toky chovají a jejich optimalizací můžeme tyto datové toky přizpůsobit a zlepšit vlastnosti jejich přenosu. Musíme si pamatovat, že toto zlepšení vlastností probíhá na úkor jiných datových toků. Proto si nejprve musíme stanovit, které z těchto parametrů jsou pro daný datový tok důležité a které méně. Můžeme mít datové toky ve kterých je důležitý pouze jeden nebo dva parametry a zbylé parametry jsou pro tento typ toku nepodstatné. Ostatní parametry můžeme využít při řešení jiného toku probíhajícího zároveň. Někdy ovšem mohou být potřebné skoro všechny z těchto parametrů pro jistý datový tok. V takovém případě si musíme rozvrhnout, které služby jsou pro nás kritické a důležité, a naopak které jsou pro nás nepodstatné nebo jen málo podstatné. Zde pak upřednostníme toky dat kritických služeb na úkor nepotřebných. Důležité je si uvědomit to, že když parametry jednoho datového toku vylepšíme, tak naopak jiný datový tok pak může v těchto parametrech ztrácet.[5, 6]

2.2.1 Přenosová rychlost - propustnost

V angličtině se tento parametr skrývá pod pojmem **bandwidth** nebo-li šířka pásma. Jedná se o množství dat, které můžeme na dané lince přenést v jednom okamžiku. Jestliže se nám cesta k cíli, kde data přenášíme, skládá z více druhů linek, tak pak je tento parametr pouze tak velký, jako je přenosová rychlost - šířka pásma té nejpomalejší linky v dané trase. Tento parametr pro jistý datový tok můžeme zvýšit tak, že nahradíme tuto linku linkou s větší propustností a nebo použijeme řešení QoS. V rámci QoS řešení pak můžeme tuto protustnost zvýšit tím, že zasílaná data zkomprimujeme nebo přizpůsobíme frontu pro daný typ dat na úkor jiného.

2.2.2 Zpoždění

Z anglického **delay**, je parametr, který nám říká kolik času je potřeba, abychom přenesli jeden paket od zdroje k cíli. Toto zpoždění je výsledkem různých dílčích zpoždění, která se nacházejí na celé trase od zdroje k cíli a v závislosti na těchto dílčích zpožděních se celkové zpoždění na jednotlivých trasách může lišit.

Prvním z těchto dílčích zpoždění je zpoždění vznikající vysláním paketu do přenosového média neboli **serializační**. Toto zpoždění závisí na velikosti paketu, který se vysílá na médium. Čím je paket větší, tím déle trvá jeho vyslání na médium.

Dalším druhem zpoždění je zpoždění vzniklé **přenosem paketu přes médium**. Zde platí, že čím delší je trasa přes médium, tím větší je toto zpoždění.

Zpoždění nám také narůstá díky **času potřebnému pro zpracování** tohoto paketu jednotlivými síťovými zařízeními. Zde se velikost zpoždění liší podle toho, jak hodně musí zařízení s paketem pracovat. Jestliže zařízení přijatý paket pouze směruje je toto zpoždění nepatrné, ale v případech, kdy zařízení musí paket analyzovat měnit jeho záhlaví a nebo daný paket šifrovat, tak může dojít k narůstu tohoto zpoždění.

Nejvíce ovlivňujícím parametrem zpoždění je **doba čekání paketů ve frontách**. Zpoždění vznikající tímto čekáním je velmi proměnlivé a závisí na množství datových toků probíhajících v dané síti. Čím více je síť využívána, tím více dat je přes ni přenášeno, a tím více jsou kladeny nároky na jednotlivá zařízení v síti. V těchto případech mohou být některá zařízení přetížena množstvím paketů, které mají zpracovat, a proto se tyto pakety, které si musí počkat na zpracování daným zařízením, ukládají do front, kde čekají až si je zařízení převezme k dalšímu zpracování.

Speciálními případy zpoždění jsou například **kodekové zpoždění** vznikající zpracováním daným kodekem a **kompresní** vznikající při kompresi dat před vysláním. V případech kdy máme na trase síť u které nevíme jak se zachová, jelikož je pod správou jiného subjektu, hovoříme o tak zvaném **síťovém zpoždění**.

Posledním dílčím zpožděním je **zpoždění vzniklé tvarovaním provozu**. Jedná se o případ, kdy zařízení přijímající data je pomalejší ve zpracování, než zařízení které data vysílá. Zde je pak vysílajícímu zařízení signalizováno, aby data zasílal pomaleji v určitých intervalech, a tím dochází ke zpoždění.

Zpoždění můžeme ovlivnit použitím koprese dat nebo přizpůsobením fronty jako u propustnosti. Zpoždění můžeme zmírnit pomocí prokládání paketů, kdy vkládáme malé pakety mezi ty velké, které nám zabírají šířku pásma při přenosu.

2.2.3 Variabilita zpoždění

Anglicky **jitter**, jedná se o parametr, který se projevuje při přímém přenosu, tedy přenosu v reálném čase, jako je například streamování videa nebo přenosu hlasu. U přenosu hlasu se jedná o VoIP telefonii. Tento parametr určuje zpoždění, které vzniká mezi příchody jednotlivých paketů do cíle. V případě že je toto zpoždění vysoké, dochází k poruchám v obrazu při přenosu a nebo k výpadkům hlasu během telefonování. Aby se tyto poruchy minimalizovaly, tak je žadoucí, aby pakety přicházely se stejným časovým odstupem a aby nedocházelo k jejich ztrátám.

2.2.4 Ztrátovost paketů

Ztrátovost je parametr, který nám říká, kolik paketů nedorazilo do cíle díky poruchám při přenosu nebo díky zahazování paketů. V rámci QoS se primárně řeší cílené zahazování, kdy ovlivňujeme, které pakety se mají zahodit a které mají zůstat a být zpracovány. Takto můžeme v případě přeplnění front ovlivnit, jaké pakety jsou prioritní a ty upřednostnit před nepodstatnými, které zahodíme.

Ztrátovost můžeme ovlivnit tím, že budeme tvarovat provoz na úkor zpoždění nebo pomocí RED nástroje, který nám může preventivně zahazovat pakety, když hrozí přeplnění front. Tím se sníží velikost tcp okna pro vysílání a dojde k pomalejšímu zasílání dat zdrojem, avšak tento mechanismus lze použít pouze pro TCP přenosy. Další možností je zvýšení velikosti front ve kterých se pakety řadí před zpracováním.

2.2.5 Doručení paketů mimo pořadí

Posledním případem který QoS řeší je, že pakety nepříjdou v pořadí v jakém byly vyslány, ale v náhodném pořadí. Toto náhodné pořadí, ve kterém pakety přicházejí, bývá způsobeno tím, že každý paket je do sítě vysílán samostatně a každý může být do cíle poslán po jiné trase, jelikož IP síť není založená na vytváření cesty od zdroje k cíli, ale je založená na přepínání paketů.

2.3 Akce prováděné pro zajištění kvality služby

Abychom zajistili patřičnou kvalitu služby, musíme si provoz nejdříve rozlišit a teprve poté můžeme nasadit jednotlivé mechanismy, které nám zaručí požadovanou kvalitu v datových tocích. Pro implementaci QoS musíme splnit určité postupy, které nám ve výsledku zajistí naši požadovanou kvalitu služby. Zde rozlišujeme celkem šest oblastí.[5, 6]

2.3.1 Oblasti QoS

1. **Klasifikace a označení (Classification and Marking)**: Slouží k rozlišení provozu.

2. **Dohlížení a tvarování provozu (Shaping and Policing)**: Provádí správu datového toku tím, že určité datové toky omezí a jiným zase vyhradí více zdrojů.

3. **Managment zahlcení (Queuing)**: Spravuje datové toky tím, že data seřazuje do front v případě přetížení zařízení.

4. **Managment vyhnutí se přetížení (Congestion Avoidance)**: Spravuje datové toky tak, že se preventivně zahazují některé pakety, aby se předcházelo přetížení na zařízeních.

5. **Signalizace (Signaling)**: Používá se u Integrated services, kdy se vyjednává spojení mezi zdrojem a cílem pomocí RSVP protokolu.

6. **Managment efektivity spoje (Link Efficiency Managment)**: Spravuje datové toky na pomalých linkách tím, že se snaží co nejefektivněji tuto linku využít, například použitím komprese.

V rámci správy provozu se nejprve pakety roztřiďují - klasifikují, poté je jim přiřazena politika, která rozhoduje, jak s nimi bude zacházeno a kolik zdrojů těmto paketům bude vyhrazeno. Tato správa by měla být prováděna co nejblíže zdrojům, aby se zajistilo, že s pakety bude zacházeno podle dané politiky při průchodu celou sítí. Jednotlivé oblasti QoS můžeme aplikovat na vstupu a jiné zase na výstupu.

2.3.2 Oblasti užívané na vstupu

1. **Classifying**: Rozlišuje typ provozu pomocí ACL nebo konfigurace a přiřazuje QoS hlavičku.

2. **Policing**: Provádí kontrolu profilu a na základě profilu určuje množství poskytovaných zdrojů.

3. **Marking**: Vyhodnotí údaje z poskytnutých zdrojů a podle profilu rozhodne, jak naložit s pakety.

4. **Queuing**: Vyhodnotí hodnotu v QoS hlavičce v poli CoS nebo DSCP a zařadí paket do vstupní fronty. V případě přetížení zařízení nám algortitmus WTD pakety zahodí.

5. **Scheduling**: Tvoří fronty s určitou váhou, kdy pakety jsou do těchto front zařazovány na základě této váhy.

2.3.3 Oblasti užívané na výstupu

1. **Queuing**: Stejně jako u vstupu, tak i na výstupu se určuje na základě QoS hlavičky do jaké výstupní fronty se paket zařadí. V případě, že jsou pakety posílány z více vstupních portů na jeden výstupní, tak se také zde využívá WTD algoritmus, který pakety při přetečení front zahazuje.

2. **Scheduling**: Plánování se používá také i na výstupních frontách. Zpravidla exituje jedna expresní fronta, kde se upřednostňují data a několik dalších front, kde se pakety zpracují podle váhy této fronty.

3 Prostředí onePK

OnePK je virtualizované prostředí na bázi linuxu s nástroji a skripty pro vývoj softwarů využívajících API v operačním systému IOS na Cisco zařízeních. V následující sekci si popíšeme, které nástroje toto prostředí obsahuje, a kde se nachází struktura jejich adresářů a souborů. Modifikací jednotlivých skriptů v tomto prostředí si můžeme vytvořit vlastní virtuální síť, kde můžeme testovat námi vyvíjené programy. Při prvním spuštění virtuálního prostředí se nám objeví okna, která se nás dotážou na hesla, která budeme používat pro vstup do tohoto prostředí a dále hesla, pomocí kterých se budeme připojovat na jednotlivá virtualizovaná zařízení přes API rozhraní.

1. **Prvním programem, který budeme používat, je virtualizační nástroj.**

   `/usr/bin/vmcloud`

   Tento nástroj se nazývá **Cisco Network Device Emulator Orchestration Tool** a použijeme jej pro vytvoření virtuální topologie na základě námi vytvořených skriptů.

2. **Dalším potřebným souborem je systém IOS, který se nám bude spouštět na jednotlivých virtuálních směrovačích.**

   `/usr/share/vmcloud/data/images/vios.ova`

   Tento soubor obsahuje celý systém IOS podporující API rozhraní, který budeme zavádět do síťových zařízení vytvářejících naši topologii na základě našich skriptů.

3. **Prostředí obsahuje také vzorovou topologii s ukázkami, jak vytvořit virtuální síť.**

   `/usr/share/vmcloud/data/examples/...`

   V tomto adresáři se nacházejí vzorové soubory na základě kterých se vytváří topologie, a které určují, jak budou zařízení mezi sebou propojena.

4. **Prostředí obsahuje vzorové kódy a programy, jejichž funkčnost si můžeme ověřit na vzorové topologii.**

   `/usr/share/vmcloud/data/examples/...`

Adresář je dále rozdělen do sekcí v závislosti na programovacím jazyku, který můžeme pro vytvoření našich programů použít. Nacházejí se zde vzory kódu využívající API pro zjištění informací ze zařízení nebo vzory, jak aplikovat určité nastavení přes tato API. Při spuštění vzorového kódu musíme zadat heslo pro směrovač na kterém kód spustíme.

5. **Prostředí také obsahuje základní konfigurační soubor.**

   `/etc/vmcloud/vmcloudrc`

   Tento soubor nám určuje místo, kde se budou nacházet dočasné soubory simulujicí topologii a konfigurace pro jednotlivé směrovače. Také jsou zde definovány porty, přes které budou virtuální směrovače přijímat konfigurační příkazy.

6. **Posledním významným adresářem je skrytý adresář, který obsahuje spouštěcí skripty pro vytvoření vzorové topologie.**

   `/home/cisco/.3node/`

   V tomto adresáři můžeme nalézt skripty, pomocí kterých se vzorová topologie vytváří. Adresář obsahuje odkazy na soubory s návrhem topologie, IOS systémem, a také odkazy na soubory pro vytvoření certifákátu u jednotlivých virtuálních směrovačů. Modifikací těchto souborů si můžeme vytvořit vlastní topologii.

4 Virtuální topologie

Spuštění topologie můžeme provést dvěma způsoby. Můžeme použít konzolové příkazy a nebo si můžeme vytvořit vlastní skripty na základě vzoru již vytvořených skriptů pro vzorovou topologii. V takovém případě bude stačit spustit zástupce, kterého si pro tuto topologii vytvoříme, a který topologii vytvoří podle nadefinovaných skriptů a nahraje potřebné certifikáty a konfigurační soubory pro jednotlivé směrovače. Toto řešení je náročnější na úpravu skriptů, ale ušetří nám čas při spouštění topologie.

4.1 Vytvoření skriptů pro spuštění

Při spuštění virtuálního prostředí zjistíme, že plocha již obsahuje zástupce pro spuštění, modifikaci a zastavení vzorové topologie. Najdeme zde zástupce pro program Eclipse, který je určen pro programování našich programů, které budou komunikovat s API rozhraním systému IOS a zástupce Create CA a Create Truststore, kteří odkazují na skripty pro vytvoření certifikátů a klíčů k virtuálním zařízením. Kromě zástupců zde můžeme nalézt i samotnou dokumentaci, která nám popisuje, jak se topologie vytváří a jaká API jsou v dané verzi onePK podporována.

4.1.1 Příprava souborů

Pro vytvoření vlastního testovacího prostředí si budeme muset připravit dva zástupce. Jedním z nich bude Start xnode, který nám spustí naši topologii a vytvoří soubory reprezentující jednotlivá síťová zařízení do kterých budeme nahrávat systém IOS spolu s konfiguracemi. Druhým zástupcem bude Stop xnode, který nám bude tolologii vypínat a mazat dočasné soubory pro uvlonění zdrojů v systému po našem testování. V rámci spuštění topologie si musíme uvědomit, že si každé virtuální zařízení rezervuje přibližně 512 MB paměti RAM pro svoji činnost.

1. **Vytvoření a úprava zástupců na ploše.**

   Nejprve si vytvoříme kopii zástupců Start 3node a Stop 3node. Tyto kopie můžeme vytvořit přes grafické prostředí a přejmenovat je na Start xnode a Stop xnode. Vstoupíme do vlastností těchto zástupců a v položce **Properties** na řádku **Command** změníme cestu k souboru.

   V zástupci Start xnode přepíšeme cestu z:

   ```
   /home/cisco/.3node/3node.py
   na
   /home/cisco/xnode/xnode.py
   ```

   V zástupci Stop xnode přepíšeme cestu z:

```
/home/cisco/.3node/3node_delete
na
/home/cisco/xnode/xnode_delete
```

Když máme zástupce na ploše vytvořené, tak budeme potřebovat samotné skripty, které nám budou topologii vytvářet.

2. **Zkopírování skriptů.**

   Skripty které budeme potřebovat se nacházejí ve skrytém adresáři. Nejprve si je zkopírujeme a přepíšeme jejich názvy.

   Zkopírování provedeme příkazem:

   ```
   cp -r /home/cisco/.3node/ /home/cisco/xnode
   ```

   Při zkopírovaní adresáře bychom měli vidět celkem devět souborů v naší nové složce s názvem xnode.

   ```
   change_subnet.sh
   gateway
   netlist.sh
   restart_tftp
   3node.py
   3node_certify.sh
   3node_create.sh
   3node_delete
   3node_info.sh
   ```

   Těmto souborům přepíšeme názvy, aby nedocházelo ke konfliktům s jinou topologii. Názvy těchto souborů po přejmenování budou tyto.

   ```
   change_subnet.sh
   gateway
   netlist.sh
   restart_tftp
   xnode.py
   xnode_certify.sh
   xnode_create.sh
   xnode_delete
   xnode_info.sh
   ```

4.1.2 Spouštěcí skripty

Když máme skripty přejmenované, můžeme začít s jejich úpravou pro naši vlastní topologii. Následující texty nám budou popisovat účel jednotlivých skriptů a změny, které musí uživatel v těchto skriptech provést, aby mohl spouštět vlastní topologii.

1. **xnode.py**

   Tento skript je první na který nás odkazuje náš zástupce Start xnode. Úkolem tohoto skriptu je ověřit, zda je naše topologie již vytvořená a stačí ji pouze spustit, nebo ještě topologie neexistuje a je ji třeba vytvořit znovu. Skript obsahuje odkazy na skript pro vytvoření topologie a skript přiřazující certifikáty pro naše virtuální směrovače. Také obsahuje funkci pro parsování vstupních dat na základě toho, zda topologii spouštíme přes konzoli nebo přes naše zástupce. V tomto skriptu musíme přepsat cesty k adresářům spolu s informacemi, že se nejedná o 3node topologii, ale o naši xnode topologii. Proto přepíšeme cesty k našim zkopírovaným skriptům z adresáře /home/cisco/.3node/... na /home/cisco/xnode/..., který nebudeme mít skrytý kvůli lepšímu přístupu k souborům.

   Upravený skript můžeme vidět v příloze B.1.

2. **xnode_certify.sh**

   Tento skript nám vytváří klíče pro jednotlivá virtuální zařízení z vytvořeného certifikátu. Obsahuje odkaz na skript createNEp.12sh, který vytvoří klíče pro jednotlivé směrovače, a poté je uloží do souboru router[x].p12 v závislosti o který směrovač se jedná. Také v tomto skriptu můžeme vidět přiřazení dns domény spolu s ověřovacím heslem, které jsme si všude nastavili jako **cisco**. Samozřejmostí je také přiřazení IP adres přes které bude náš program se směrovači komunikovat. Poté v samotném skriptu nastavujeme cesty přes odkazy, kdy cesta k createNEp.12sh skriptu je /home/cisco/.simpleCA/createNEp.12sh.

   Výsledný skript po úpravách můžeme vidět v příloze B.2.

3. **xnode_create.sh**

   Tento skript je hlavní při spouštění virtuální topologie. Skript obsahuje odkaz na soubor **xnode.virl**, který obsahuje námi nakonfigurovanou topologii. Je zde obsažen samotný spouštěcí příkaz k vytvoření topologie a vytvoření oznamovací tabulky s průběhem spouštění. Také nás informuje o úspěšném nebo neúspěšném vytvoření topologie.

   Skript po úpravách můžeme vidět v příloze B.3.

4. **xnode_delete**

   Jak vyplývá z názvu skriptu, tak skript delete má za úkol zastavit procesy simulující virtuální topologii a vymazat soubory, které ji tvoří. Při spuštění dojde k zastavení procesů a k vymazání konfigurací a dočasných souborů, které vznikají při vytváření topologie ve skrytém adresáři /var/vmcloud/.vmCloud_cisco. V tomto adresáři zůstanou jako jediné soubory s názvem vmlog.log, který se přepisuje a obsahuje informace o právě spuštěných topologiích. Dalším souborem je xnodevmlog.log, který vytváříme pomocí skriptu create. Tento soubor se také přepisuje a obsahuje informace o jednotlivých směrovačích a jejich aktuálních portech přes které se na směrovače můžeme připojit, například přes telnet.

   Celý skript můžeme vidět v příloze B.4.

5. **xnode_info.sh**

   Tento skript pouze definuje parametry informačního okna a odkazuje se na informační log soubor /var/vmcloud/.vmCloud_cisco/xnodevmlog.log, který obsahuje informace s IP adresami a porty pro připojení a management jednotlivých virtuálních směrovačů.

   Skript je krátký a je obsažen v příloze B.5.

6. **netlist.sh**

   Podobným skriptem jako je xnode_info.sh je skript netlist.sh. Tento skript je také informační a obsahuje informace o rozložení okna a odkaz na druhý log soubor. Oproti předchozímu skriptu tento skript obsahuje informace s právě spuštěnými topologiemi. Jinak se jedná o téměř podobný skript.

   Celý skript můžeme vidět v příloze B.6.

4.2 Vytvoření topologie

Když máme vytvořené všechny skripty, které nám spouštějí topologii, tak si můžeme připravit soubor se samotnou topologií, pro kterou chceme management provádět a na které budeme testovat naše programy využívající API systému IOS. Soubor na základě kterého bude topologie vytvářena se bude jmenovat **xnode.vir**l a jeho umístění bude v adresáři /home/cisco/vmcloud-example-networks/xnode/, který jsme si nastavili ve skriptu **create.sh**. V rámci topologie jsem si zvolil propojení šesti směrovačů.

Při spuštění topologie zástupcem start xnode dojde k vytvoření jednotlivých směrovačů spolu s vytvořením jednotlivých rozhraní, díky kterým se můžeme do této topologie připojit. Samotný soubor obsahuje propojení jednotlivých směrovačů mezi sebou na předem definovaných rozhraních, jako by se jednalo o typicky fyzické propojení UTP kabeláží. Soubor je tvořen určitou

terminologií, kterou při vytváření topologie musíme dodržet, aby nám výsledná virtuální topologie fungovala.

Jak bylo řečeno dříve, tak adresář, který bude obsahovat všechny soubory se skripty k dané topologii, bude /home/cisco/vmcloud-example-networks/xnode/. Hlavní soubor je zde xnode.virl, který obsahuje informace o celé topologii a je hlavním při sestavení topologie. Kromě tohoto souboru si můžeme ve složce vytvořit také soubory s konfiguracemi jednotlivých směrovačů, takže při spuštění topologie a spuštění systému IOS na směrovačích se nám také nahrají tyto konfigurace. Při prvním spuštění topologie se v této složce vytvoří soubory s klíči pro přístup našich aplikací a programů k jednotlivým směrovačům. Tyto soubory jsou ve formátu název směrovače.p12.

1. **xnode.virl**

   Soubor je ve formátu xml a obsahuje několik parametrů, které nám vytvářejí celou topologii. Prvním parametrem je parametr <node>, který nám definuje samotné směrovače nebo přepínací prvky. U tohoto parametru si určíme název prvku, typ prvku, umístění a cestu k IOS. Prvky můžeme definovat jako typ SIMPLE, kdy se prvek chová jako směrovač nebo SEGMENT, který nám propojí několik prvků mezi sebou jako přepínač a nebo ASSET, který slouží k vytvoření vazby mezi fyzickým a virtuálním rozhraním, abychom mohli topologii spojit s reálnou sítí.

   ```
   <node name="router5" type="SIMPLE" subtype="vios" location=
   "100,500" vmImage="\newline /usr/share/vmcloud/data/images/
   vios.ova">
   ```

   Dalším parametrem je <entry key>. Pomocí tohoto parametru definujeme vstupní hodnoty pro naše zařízení. Mezi tyto hodnoty patří cesta k startup konfiguraci a k certifikátům nutným pro inicializaci spojení mezi aplikacemi a směrovači.

   ```
   <entry key="bootstrap configuration" type="String">/home/
   cisco/vmcloud-example-networks/xnode/router1.con</entry>
   <entry key="import files" type="String">/home/cisco/vmcloud-
   example-networks/\newline xnode/router1.p12</entry>
   ```

   Uvnitř segmentů definujeme počet a názvy jednotlivých rozhraní, která budou jednotlivé síťové prvky propojovat. To provedeme pomocí parametru <interface>.

   ```
   <interface name="GigabitEthernet0/0"/>
   <interface name="GigabitEthernet0/1"/>
   ```

Když máme všechny síťové prvky nastaveny podle našich představ, tak můžeme tyto prvky navzájem mezi sebou propojit pomocí parametru <connection>. Tento parametr nám reprezentuje jednotlivé linky mezi prvky, jako by se jednalo o spojení ethernetovým kabelem. Při sestavování spojů si musíme uvědomit, který prvek má jaké pořadí a jaké rozhraní chceme se kterým propojit. Například pokud si segment nazveme <node name="lan_ex_2, neznamená to, že by tento segment byl node[2], ale jelikož byl vytvořen až sedmý v pořadí je tento segment jako node[7]. Stejné chování platí pro jendnotlivá rozhraní. Ikdyž jsme si nadefinovali rozhraní <interface name="GigabitEthernet0/0"/>, neznamená to, že rozhraní je interface[0], ale jedná se o rozhraní interface[1].

```
<connection src="/topology/node[1]/interface[1]" dst="/topology/node[10]
/inter-face[1]"/>
```

Celý soubor s topologii můžeme vidět v příloze B.7.

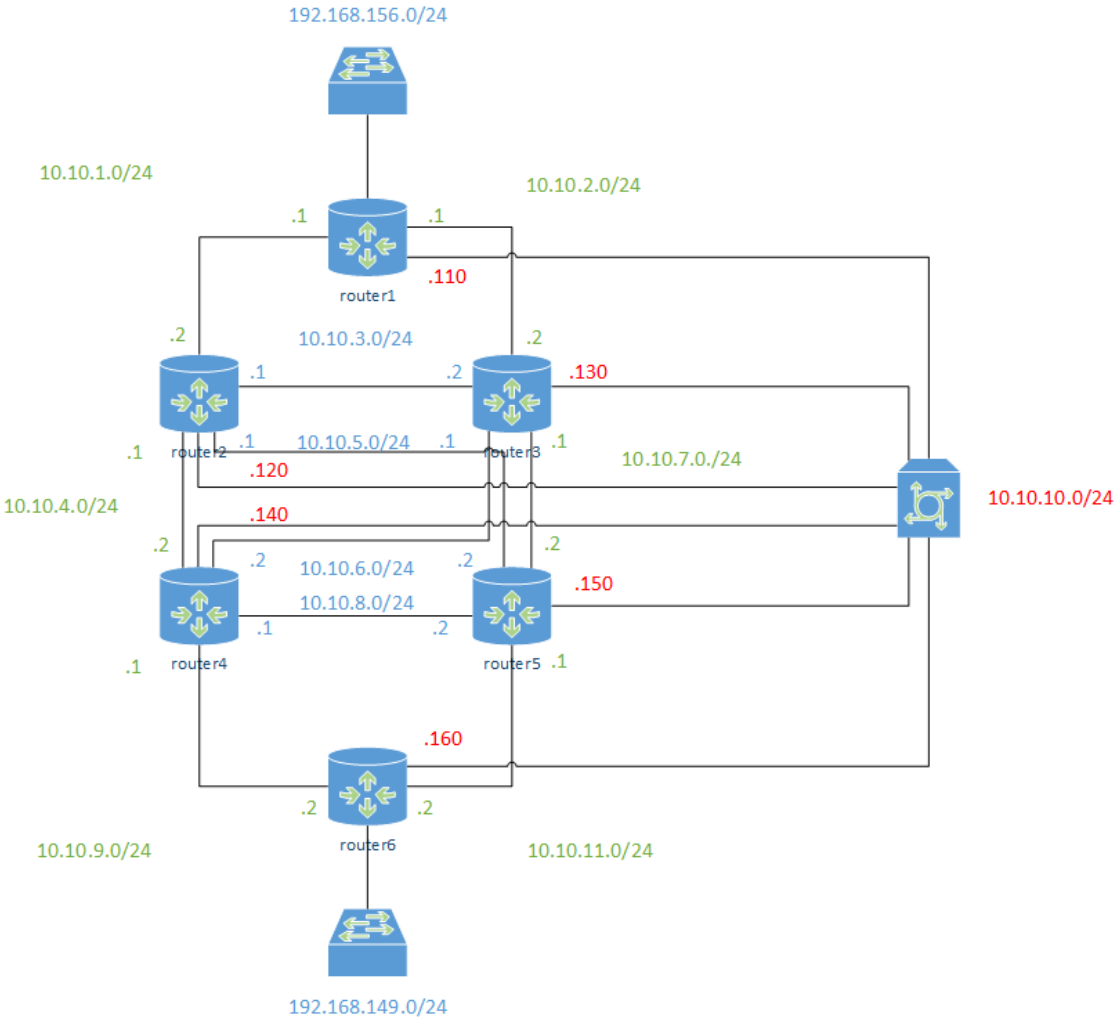
2. **router[x].con**

   Druhým souborem při tvoření topologie je konfigurace jednotlivých směrovačů, která bude aplikována při spuštění topologie. Tuto konfiguraci musíme definovat pro každý vytvořený síťový prvek, pokud nechceme mít tyto prvky bez konfigurace.

   Ukázku konfigurace můžeme nalézt v příloze B.8.

4.2.1 Fyzické propojení prvků v topologii

Po spuštění souboru xnode.virl se nám vytvoří jednotlivé prvky, které budou mezi sebou propojeny a komunikovat jako na obrázku 4.1. Síť obsahuje šest vzájemně propojených směrovačů, které jsou zakončeny pomocí přepínačů. Síť na straně směrovače pod názvem router1 je napojena do rozhraní, které je pomocí NAT připojeno do fyzického prostředí. Síť za směrovačem pod názvem router6 je pak napojena na virtuální rozhraní linuxového systému, který nám tuto síť virtualizuje. Celá síť nám při spuštění pracuje pomocí protokolu OSPF spolu s přednastavenými subnety. V rámci nastavení jsou také na směrovačích nastaveny DNS konfigurace pro otestování připojení k reálné síti.



Obrázek 4.1: Fyzické propojení prvků v síti.

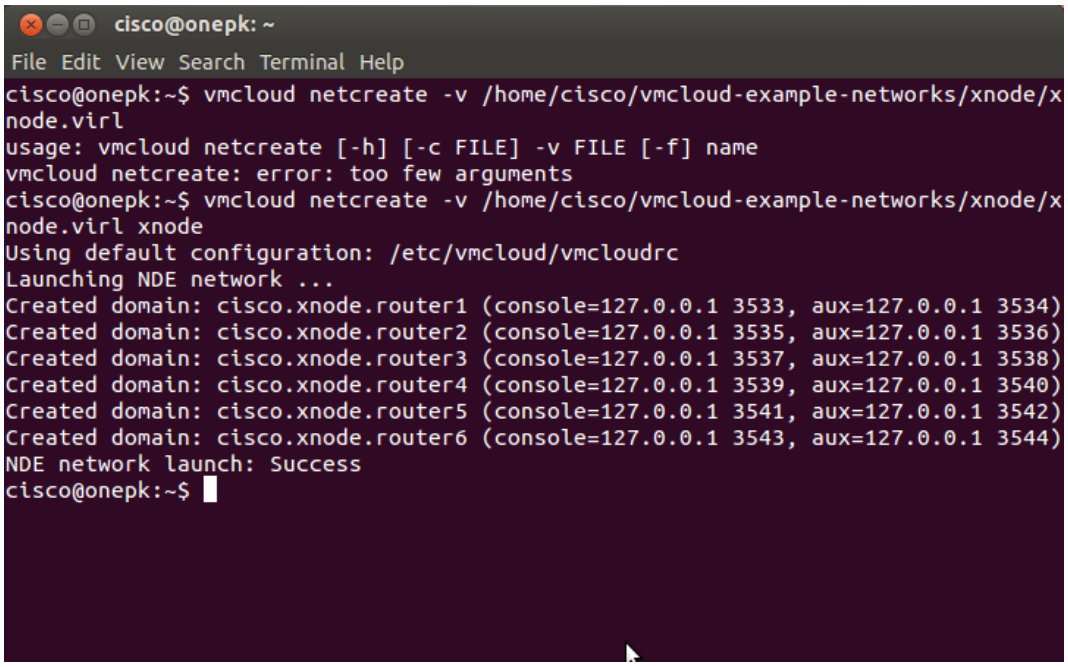
4.3 Spuštění topologie

Když máme všechny skripty připraveny, můžeme topologii spustit pomocí **start zástupce**. Avšak pokud bychom si tuto topologii nepřipravili podle skriptů, můžeme ji spustit také manuálně přes příkazy v konzoli. V takovém případě potřebujeme mít připravený jen soubor **xnode.virl**. Celou topologii spustíme příkazem.

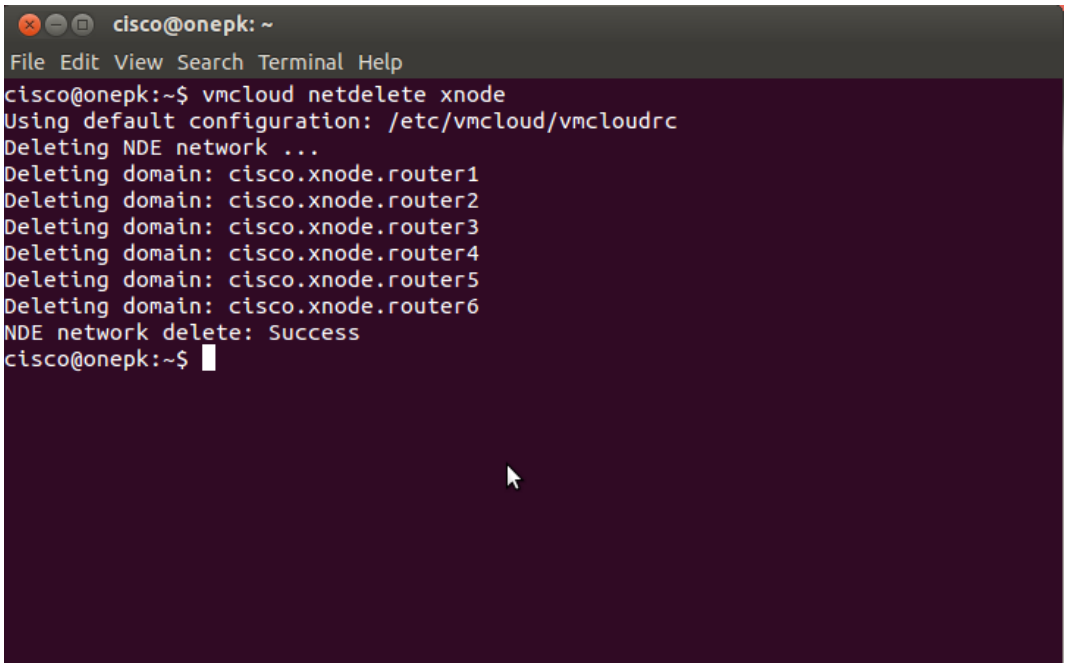
```
vmcloud netcreate -v /home/cisco/vmcloud-example-networks/xnode/xnode.virl xnode
```

Poté se můžeme připojit k jednotlivým směrovačům přes telnet a port, který se nám zobrazí ve výpisu při vytváření této topologie. Pokud budeme chtít topologii ukončit můžeme to provést příkazem.

```
vmcloud netdelete xnode
```

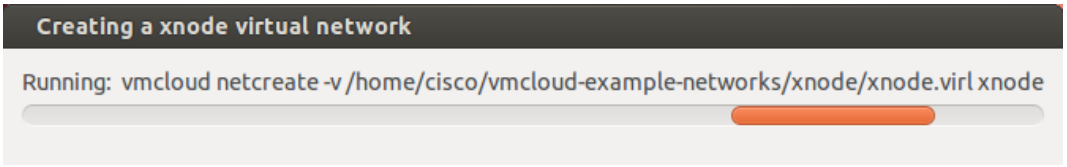


Obrázek 4.2: Vytvoření topologie přes konzolový příkaz.



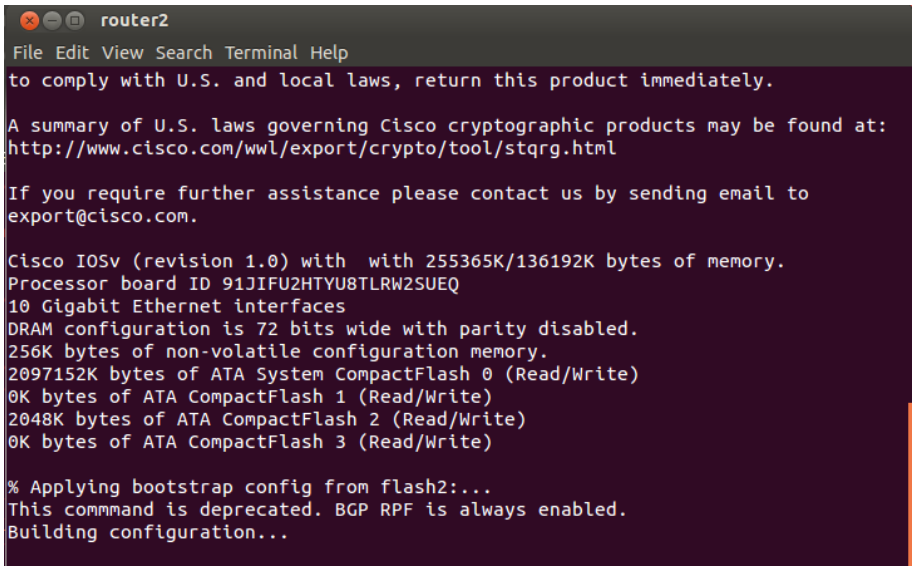
Obrázek 4.3: Ukončení topologie přes konzolový příkaz.

Druhý způsob jak spustit topologii je přes našeho start zástupce. Zde se nám budou objevovat dialogová okna, která nás budou informovat o průběhu spouštění topologie. Okamžitě po spuštění se nám objeví dialogové okno oznamující vytváření souborů emulujících naši topologii. Při tomto procesu se nám budou vytvářet soubory reprezentující jednotlivé prvky ve skrytém adresáři **/var/vmcloud/.vmCloud_cisco**.



Obrázek 4.4: Dialogové okno průběhu spouštění.

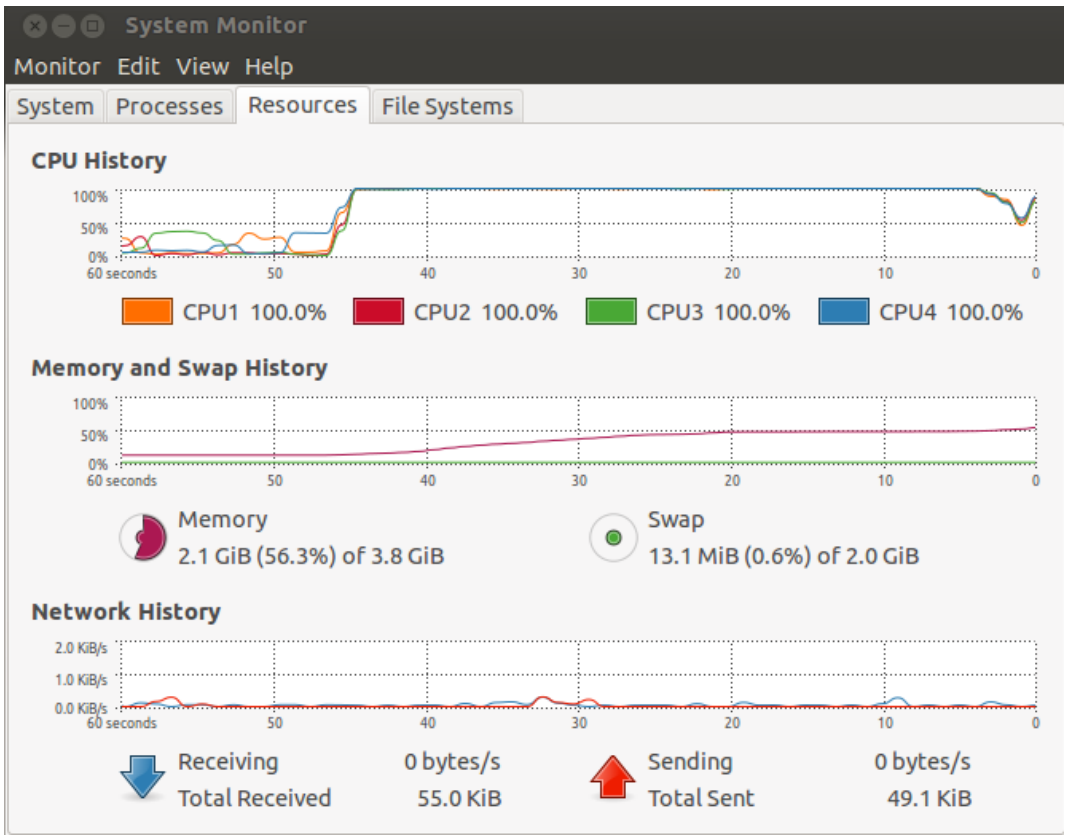
Pokud máme vše nastaveno správně, objeví se hláška o úspěšném vytvoření sítě. Dojede k automatickému spuštění terminálových oken pro každý směrovač v topologii a my budeme moci vidět průběh zavádění systému IOS do těchto síťových prvků.



Obrázek 4.5: Zavádění systému do směrovače č.2.

Spuštěná topologie má znatelný vliv na vytížení zdrojů ve virtuálním prostředí. Největší vytížení je na procesoru během zavádění systému IOS. Také paměť RAM je vytížena, kdy minimální množství paměti přidělené jednomu směrovači se pohybuje kolem 500 MB. K uvolnění zdrojů dochází až při vypnutí topologie, kdy dojde k uvolnění paměti a k vymazání souborů ve skryté složce **/var/vmcloud/.vmCloud_cisco**.

Na obrázku 4.6 můžeme vidět k jakému vytížení systému dochází během zavádění systému IOS do síťových prvků.



Obrázek 4.6: Vytížení systému během zavádění systému IOS.

5 Přesměrování provozu během výskytu CRC chyb na lince

První program je zaměřený na přesměrování provozu při výskytu chyb na fyzické vrstvě, která má za úkol přenášet data ve formě impulzů přes různé prostředí. Samotný přenos na fyzické vrstvě probíhá přes různá média. Mezi standardní média se řadí přenos dat pomocí kabelového nebo bezdrátového připojení. Kabelový přenos můžeme dále rozdělit na přenos pomocí metalického kabelu nebo pomocí optického vlákna. Pro obě tato prostředí existují různé mechanismy zajišťující bezchybný přenos dat. Toho je docíleno například pomocí mechanismu, který kontroluje přenesený signál pomocí kontrolní sekvence dat, která jsou součástí přenesených dat. V případě porovnání kontrolní sekvence, kdy hodnota těchto dat souhlasí s očekávanou hodnotou, máme jistotu, že data byla přenesena správně a jsou připravená k použití. V případě že dojde k výskytu chyby, tak nám zpravidla protokoly vyšší vrstvy tuto chybu zaznamenají a zažádají o nový přenos těchto dat.

Jestliže nám tedy vznikají chyby na fyzické vrstvě, pak se data zpravidla posílají několikrát přes tuto linku, dokud není zajištěn jejich bezchybný přenos. Toto posílání dat nám však při vyšším počtu chyb může zvýšit čas odezvy a nebo snížit propustnost na lince. Operační systém IOS je schopen si zaznamenávat několik těchto typů chyb na rozhraních, které započítává do svých statistik. Díky těmto údajům můžeme následně přizpůsobovat topologii, aby přenos dat probíhal efektivněji.

5.1 Naprogramované funkce

Základní přihlašovací údaje jsou nastaveny jako uživatelské jméno cisco[x] a číslo podle směrovače na který se připojujeme. Stejné nastavení platí i pro heslo. Například když budeme chtít program spustit na směrovači s číslem 2, pak údaje k přihlášení budou tyto.[9, 10, 11, 12, 13, 14]

```
username: cisco2
password: cisco2
```

1. **Navázání spojení**

   První funkce kterou použijeme je funkce **connect**. Tato funkce má za úkol zajistit komunikaci programu se samotnými směrovači. K tomuto účelu funkce využívá API element a SessionHandle .

   ```
   public boolean connect(String applicationName)
   ```

   Aby došlo k navázání spojení potřebujeme mít k dispozici tři základní parametry.

```
username - uživatelské jméno
password - heslo pro autentizaci
sessionConf - parametry spojení
```

2. **Parametry spojení**

   API SessionConfig použijeme pro stanovení parametrů komunikace mezi programem a směrovači. Komunikace probíhá zpravidla pomocí protokolu TSL, což je protokol sloužící ke komunikaci pomocí zašifrovaných dat. Kromě protokolu můžeme nastavit i port přes který má komunikace probíhat. Jako defaultní port je port 15002 podle specifikací onePK.

   Funkce SessionConfig slouží pro vytvoření konfigurace s předem definovanými parametry.

   ```
   private SessionConfig createSessionConfig()
   ```

   Tato funkce je použita pro každý směrovač, a proto všechna spojení mají stejné parametry. Parametry které nám funkce nastavuje můžeme vidět níže.

   ```
   transportí mód - TSL
   port - defautní
   velikost fronty událostí
   maximální počet vláken pro událost
   čas pro znovu navázání spojení
   ```

   U navázáného spojení pak nastavujeme hodnoty pro zacházení se spojením.

   ```
   zahazování nových událostí, v případě že dojde k přeplnění fronty událostí
   nastavení času udržení spojení
   nastavení času mezi požadavky o udržení spojení v nečinnosti
   nastavení časového intervalu pro udržení spojení
   čas opakování požadavku pro znovu navázání spojení
   přiřazení spojení do databáze důvěryhodných spojení
   ```

3. **Důvěryhodnost spojení**

   Při každém nově navázaném spojení programu se směrovači můžeme rozhodnou o důvěryhodnosti jednotlivých spojů. K tomuto účelu nám v programu slouží další funkce showPinningDialog.

   ```
   public Decision showPinningDialog(String host, String hashType, String
   fingerprint, boolean changed)
   ```

Cílem této funkce je určit důvěryhodnost programu, který přistupuje ke směrovači skrz API. Funkce nám dává na výběr tři možnosti.

```
zakázat spojení
povolit spojení
povolit spojení a přiřadit jej do databáze důvěryhodných
```

4. **Vstupní data**

   Vstupními daty jsou IP adresy směrovačů se kterými program komunikuje. Pro tento účel jsme si vytvořili dvě funkce. První je funkce readProperties. Tato funkce získává data ze souboru **application.properties**, který obsahuje základní informace o jednotlivých směrovačích, jako je IP adresa směrovače, cesta k pinning souboru a cesta k trustStore souboru.

   ```
   private boolean readProperties()
   ```

   Druhá funkce kterou algoritmus obsahuje, slouží pro spuštění programu přes konzoli.

   ```
   private boolean parseCommandLine(String[] args)
   ```

   Cílem funkce je rozeznání vstupních parametrů zadaných do konzole a spuštění programu pro směrovače. Mezi vstupní parametry, které se zadávají do konzole jsou IP adresy směrovačů, které nastavujeme parametrem a až a[x] s číslem podle množství směrovačů pro které byl program navržen. Pomocí parametu -u zadáme uživatelské jméno prvního směrovače a parametr -p, kde napíšeme heslo pro ověření.

5. **Autentizace**

   Další funkcí v programu je funkce showAuthenticationDialog. Tato funkce nám vytváří dialogové okno pro přihlášení ke směrovačům. Funkce je tvořena dvěmi částmi. První část je dialogové okno aplikace a druhá část obsahuje dialogový zápis údajů přes konzoli.

   ```
   public void showAuthenticationDialog()
   ```

6. **Seznam rozhraní**

   Funkce getAllInterfaces má za úkol vytvořit seznam rozhraní na směrovači. Později tuto funkci v programu používáme pro definování rozhraní, na kterém budeme chtít provést nějakou akci.

```
public List<NetworkInterface> getAllInterfaces()
```

7. **Získání hodnoty CRC chyb**

   Funkce pollStatistics() je navržena pro získání aktuální hodnoty CRC na lince, kterou používáme jako primární.

   ```
   public void pollStatistics() throws OnepException, InterruptedException
   ```

8. **Přidání směrovací informace**

   Funkce getRIB má za úkol rozšířit směrovací tabulku o novou cestu, která přesměruje provoz přes jiné rozhraní, než pres které standardně provoz prochází. Abychom mohli přidat novou cestu potřebujeme dvě funkce. Funkci pro vytvoření směrování a funkci pro přidání směrovací informace.

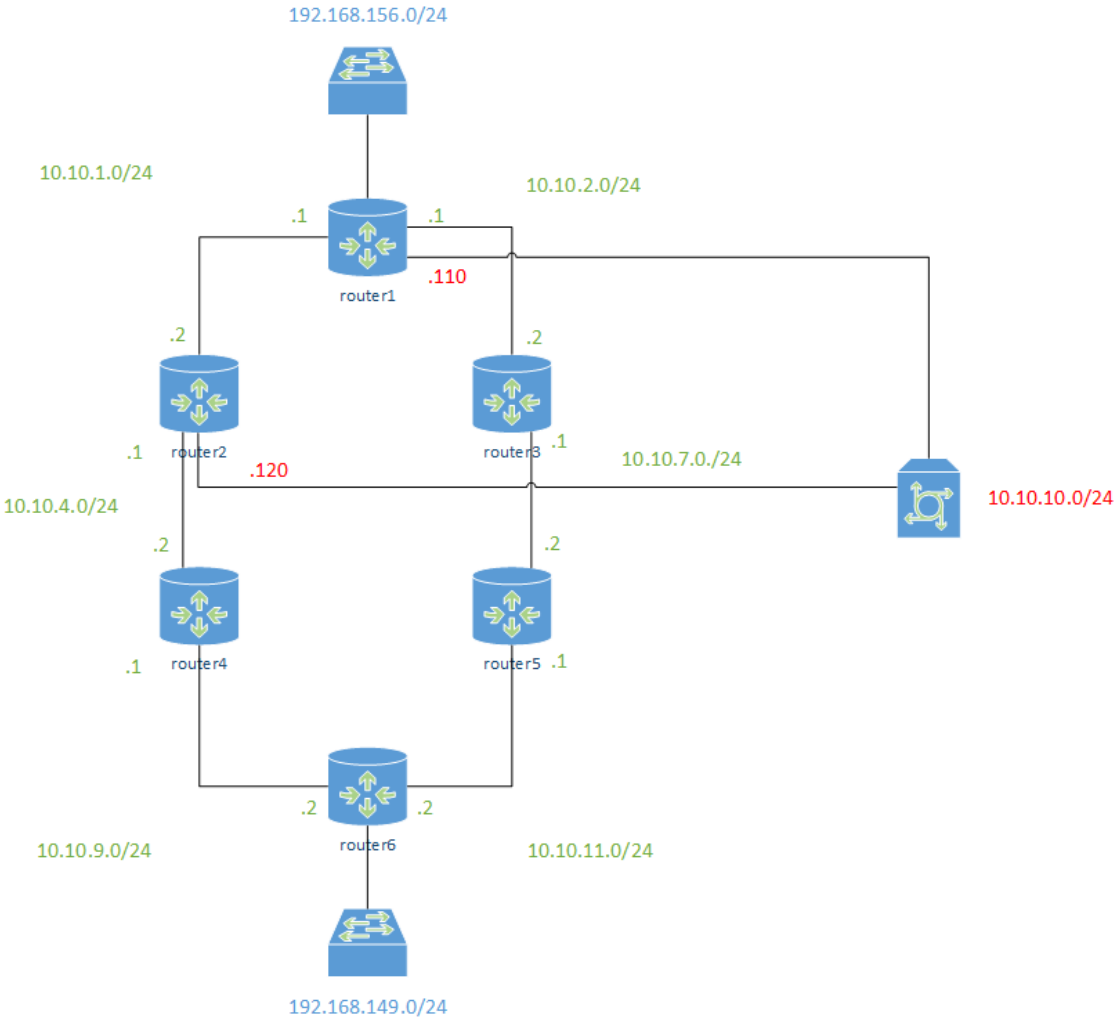
   ```
   public RIB getRIB() throws OnepConnectionException
   addRoutes(AppRouteTable appRouteTable) throws OnepConnectionException,
   OnepIllegalArgumentException, OnepRemoteProcedureException, OnepException,
   UnknownHostException
   ```

5.2 Testování programu

Než si spustíme náš program, tak si pro tento účel nejprve upravíme topologii a necháme spuštěné pouze ty linky, pro které budeme program testovat. Logické zapojení topologie pro testování můžeme vidět na obrázku 5.1. Připojíme se na směrovač router1 a vyjmeme síť 10.10.2.0/24 z OSPF procesu. Tento krok děláme, abychom si zajistili, že nám OSPF nebude ovlivňovat druhou cestu, protože přes tuto síť nám bude směrování přidávat samotný program. Také si tím zajistíme, že primární cesta pro směrování bude skrze směrovače router1. router2, router4 a router6.

```
enable
conf term
router ospf 1
no network 10.10.2.0 0.0.0.255 area 0
end
```

Jestliže budeme spuštět program přes konzoli, pak budeme muset překopírovat složku s programem z adresáře /home/cisco/onePK-sdk-1.2.0.173/java/sample-apps/src/main/java/com /cisco/onep/program1 do adresáře /opt/cisco/onep/java/sdkjava-1.2.0.173/java/sample-aps/src



Obrázek 5.1: Logické zapojení topologie při testování.

/main/java/com/cisco/onep/examples/program1, kde se nachází SDK knihovna potřebná pro kompilaci a spuštění algoritmu. Kompilaci programu provedeme tak, že se přesuneme do adresáře /opt/cisco/onep/java/sdk-java-1.2.0.173/java/sample-aps/src/main/java/ pomocí cd příkazu a spustíme příkaz.

```
javac com/cisco/onep/examples/program1/program1.java
```

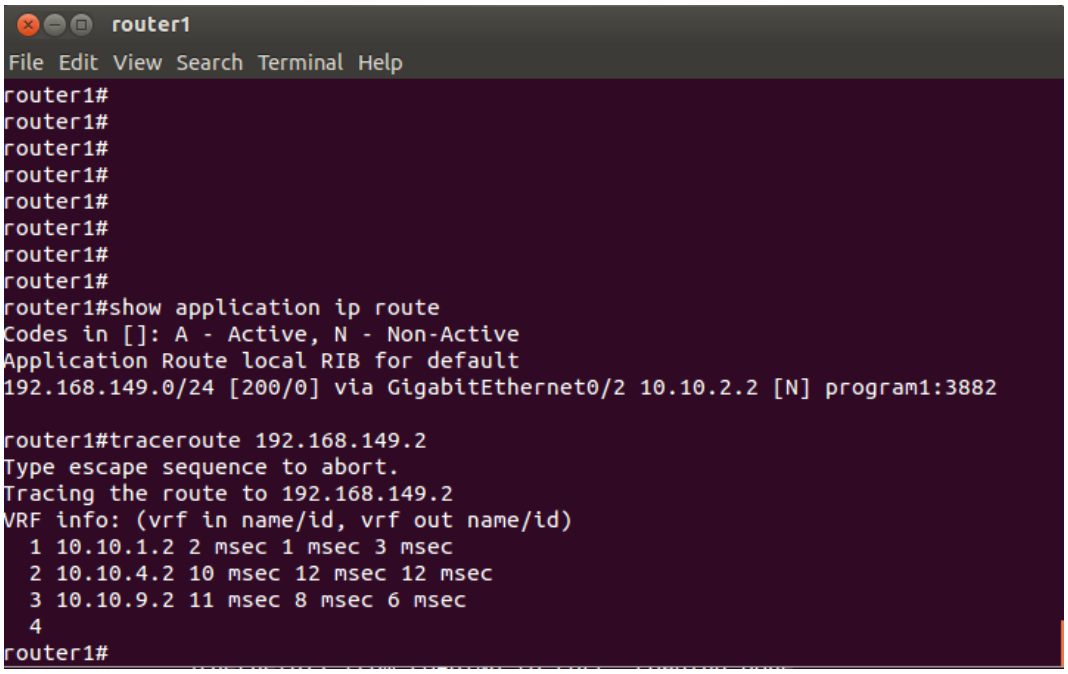
Program spustíme pomocí java příkazu.

```
java com.cisco.onep.examples.program1.program1 -a 10.10.10.110 -a2 10.10.10.120
-u cisco1 -p cisco1
```

5.3 Chování topologie při použití programu

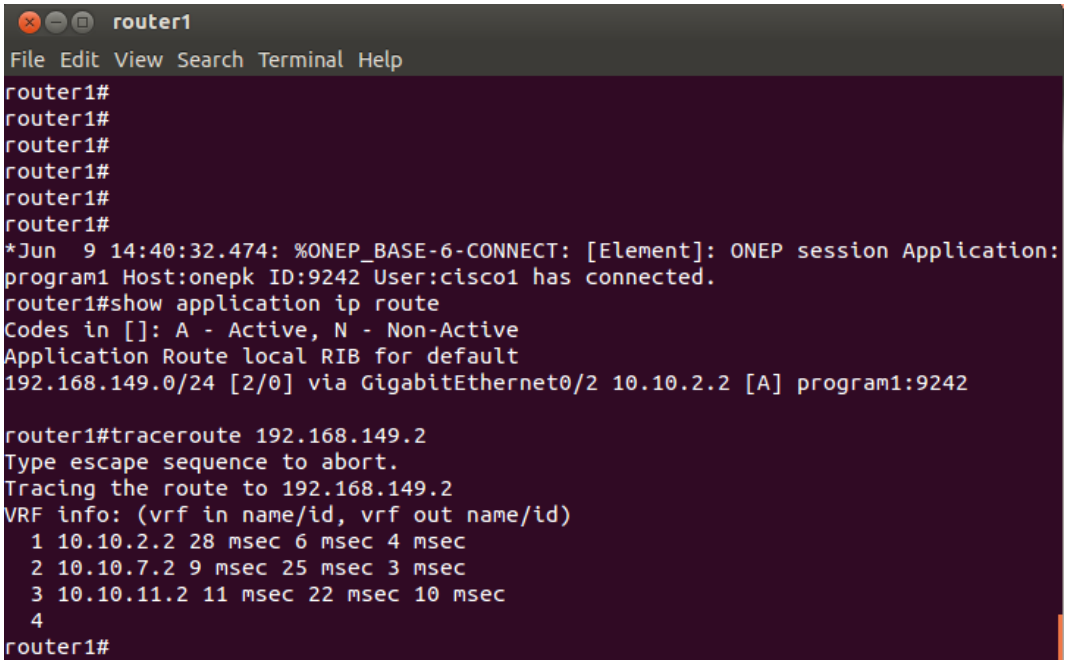
Při spuštění programu dojde k vytvoření spojení se směrovačem router1 a router2. Cílem programu je přesměrování jednosměrného provozu při výskytu chyb na lince, který přichází z internetového prostředí, tedy z natované sítě 192.168.156.0/24 směrem do sítě 192.168.149.0/24. Proto abychom mohli toto přesměrování provést budeme muset komunikovat se směrovačem router1 s adresou 10.10.10.110 a směrovačem s názvem router2 s IP adresou 10.10.10.120. Obě tyto IP adresy patří síti 10.10.10.0/24, kterou používáme pro management.

Směrovač s názvem router1 je směrovač rozhodující, kterou trasou bude provoz do sítě 192.168.149.0/24 poslán, proto nám zde program upravuje směrovací informace, kdy přidá směrování do sítě 192.168.149.0/24 a pro toto směrování mění metriku v závislosti na výskytu chyb. Pro testovací účely jsme si stanovili primární cestu přes směrovače router1, router2, router4 a router6. Cílem programu je přesměrovat provoz do této sítě přes směrovače router1, router3, router4 a router6 při výskytu chyb na námi zvolené primární trase. V případě že se na lince nevyskytují žádné CRC chyby, pak provoz prochází levou stranou topologie na obrázku 5.1. Toto chování si můžeme ověřit pomocí traceroute příkazu na směrovači router1, kdy na obrázku 5.2 můžeme vidět cestu přes jednotlivé sítě do sítě 192.168.149.0/24. Ve výpisu z tohoto obrázku můžeme také vidět přidanou směrovací informaci do sítě 192.168.149.0/24, kdy metrika je nastavena na hodnotu 200 a stav tohoto směrování je jako N, tedy směrovací informace je neaktivní.



Obrázek 5.2: Výstup ze směrovače pro linku bez fyzických chyb.

Pokud dochází k výskytu chyb na lince v příchozím směru od směrovače router1 ke směrovači router2, tak se nám přenastaví metrika v přidané směrovací informaci na hodnotu 2 a směrování je přepnuto do aktivního stavu A. Také si směrování můžeme opět prověřit pomocí traceroute příkazu. Oba tyto výpisy o změně trasy můžeme vyčíst z obrázku 5.3.



Obrázek 5.3: Výstup ze směrovače pro linku při výskytu chyb.

Samotné CRC chyby program získává ze statistik rozhraní směrovače router2 na které data přicházejí. V případě že nám chyby na rozhraní nenarůstají, pak je provoz přesměrován zpět přes původní rozhraní.

6 Změna parametrů QOS v čase

Tento program nám nastavuje QOS na směrovačích tak, že vytváří různé třídy, politiky a parametry, které nám zvýhodňují daný druh provozu. Aplikováním QOS zvýšíme efektivitu přenosu dat pro určitý provoz, ale na úkor jiného druhu provozu. Musíme si tedy předem rozvrhnout, který druh provozu potřebujeme prioritizovat a naopak, který provoz je pro nás nejméně užitečný a u kterého se umíme smířit se ztrátami dat.

Běžně se QOS nastavuje pro jednotlivé linky mezi uživatelem a poskytovatelem připojení, kde se počítá s určitým typem provozu, který může být zpoplatněn. QOS lze také aplikovat pro celou nebo část sítě. Podmínkou, aby nám QOS fungoval správně, je nastavit dané profily na všech směrovačích po cestě, které mají podle QOS pracovat. V případě, že nějaký směrovač není nastaven stejně, dochází k porušení těchto pravidel a QOS nám nemusí fungovat. Proto se primárně QOS nastavuje jen pro dané linky mezi dvěma subjekty, jako je uživatel a poskytovatel. Jestliže se rozhodneme aplikovat QOS pro danou síť, pak musíme překonfigurovat všechy směrovače po cestě. Tento proces může být nákladný a zdlouhavý u velkých sítí a výsledek nemusí být vždy podle našich představ.

Když máme QOS aplikováno, tak se může stát, že aplikované řešení není úplně ideální, jelikož v rámci jednoho prostředí toto řešení funguje dobře, ale pro jiné typy provozu zase ne, protože se podmínky v dané síti mohou změnit v závislosti na čase. Následující program nám pomůže vytvořit prostředí, kde je aplikováno QOS tak, aby došlo k nejefektivnějšímu nastavení QOS bez vnějšího zásahu.

6.1 Naprogramované funkce

Program, který nám řeší nastavení QOS podle času, používá některé stejné nebo podobné funkce jako program, který nám řešil přesměrování provozu na základě CRC chyb. Níže můžeme vidět funkce, které jsou shodné nebo s drobnými úpravami jako u předchozího programu.[9, 10, 11, 12, 13, 14]

1. **Shodné nebo upravené funkce**

```
public boolean connect(String applicationName)
private SessionConfig createSessionConfig()
public Decision showPinningDialog(String host, String hashType, String
fingerprint, boolean changed)
private boolean readProperties()
private boolean parseCommandLine(String[] args)
public void showAuthenticationDialog()
```

```
public List<NetworkInterface> getAllInterfaces()
```

Z předchozího seznamu můžeme vidět funkce, které jsou si podobné, jako v prvním programu. Funkce connect nám navazuje spojení se čtyřmi směrovači místo dvou, funkce showPinningDialog se nám spustí čtyřikrát, protože program navazuje spojení se čtyřmi směrovači. U funkce parseCommandLine máme přidány navíc parametry -a3 a -a4, kterými v konzoli definujeme další dva směrovače a u funkce showAuthenticationDialog jsme si rozšířili dialogové okno o další přihlašovací údaje.

2. **Vytvoření balíčku se službami**

   Tato nová funkce má za úkol vytvořit balíček, který obsahuje všechny služby, nastavení, třídy a jednotlivé politiky. Při vytvoření těchto služeb a různých politik nám tento balíček slouží jako kontejner, který tyto služby shromažďuje a který aplikujeme pro každý směrovač. Funkce má za úkol vytvořit balíčky s QOS parametry pro každý směrovač zvlášť. Zároveň tyto balíčky obsahují jednotlivá omezení a možnosti pro daný směrovač, které vymezují jaké služby mohou být na směrovači použity.

   ```
   public void createCapabilities() throws OnepRemoteProcedureException,
   OnepConnectionException
   ```

3. **Vytvoření tříd**

   Stejně jako u standardní konfigurace Cisco směrovače přes konzoli musíme i přes API rozhraní vytvořit jednotlivé třídy provozu, které nám budou definovat provoz procházející zařízením. Program nám vytváří tři třídy, jednu pro hlas, obrazová data a normální data. V hlavní funkci void main přiřazujeme k těmto třídám jednotlivé parametry, pomocí kterých se má provoz rozlišit.

   ```
   public void createmap() throws OnepIllegalArgumentException
   ```

4. **Vytvoření politiky**

   Funkce createpolicy nám vytváří politiku, která obsahuje jednotlivé třídy provozu podle kterých se nám provoz rozlišuje.

   ```
   public void createpolicy () throws OnepIllegalArgumentException
   ```

5. **Aplikování třídy na směrovači**

   Pro aplikování jednotlivých tříd a jejich vytvoření na směrovači potřebujeme funkci getClassMapResult, která tyto změny aplikuje. Tato funkce komunikuje s API rozhraním a zasílá informace směrovačům, jak byly jednotlivé třídy v rámci programu nastaveny.

```
public void getClassMapResult() throws OnepRemoteProcedureException,
OnepIllegalArgumentException, OnepConnectionException}
```

6. **Aplikování politiky na směrovači**

   Kromě funkce která komunikuje se směrovačem a nastavuje jednotlivé třídy, potřebujeme vytvořit politiku, která tyto třídy bude obsahovat. K tomuto účelu potřebujeme funkci getPolicyMapResult, která vytvořenou politiku aplikuje.

```
public void getPolicyMapResult() throws OnepRemoteProcedureException,
OnepIllegalArgumentException, OnepConnectionException}
```

7. **Nastavení parametrů tříd**

   Pro nastavení jednotlivých parametrů v rámci tříd které spadají pod danou politiku, potřebujeme vytvořit vstupní hodnoty ke kterým přiřadíme jejich vlastnosti, jako je propustnost, velikost front a různá omezení. Tyto parametry nám tvoří funkce createPolicymapEntry.

```
public void createPolicymapEntry() throws OnepIllegalArgumentException
```

8. **Získání aktuálního času**

   Funkce gettime nám získává aktuální systémový čas. Tento čas nám bude ovlivňovat, kdy bude která politika použita v naší síti.

```
public void gettime()}
```

9. **Výběr rozhraní**

   Funkce getSpecInterface má za úkol výběr správného rozhraní pro aplikování naší QOS politiky. Jedná se o stejný princip, jako aplikace politik na rozhraní při nastavení přes konzoli.

```
public NetworkInterface getSpecInterface0()
```

6.2 Testování programu

Před spuštěním programu nejprve zrušíme směrování přes pravou větev v síti na obrázku 5.1. To provedeme pomocí příkazu shutdown na rozhraních, které vedou na tuto pravou stranu na všech směrovačích. Tím zajistíme, že pro testování budou použity směrovače router1, router2, router4 a router6.

```
enable
conf term
interface Gigabitethernet 0/2
shutdown
end
```

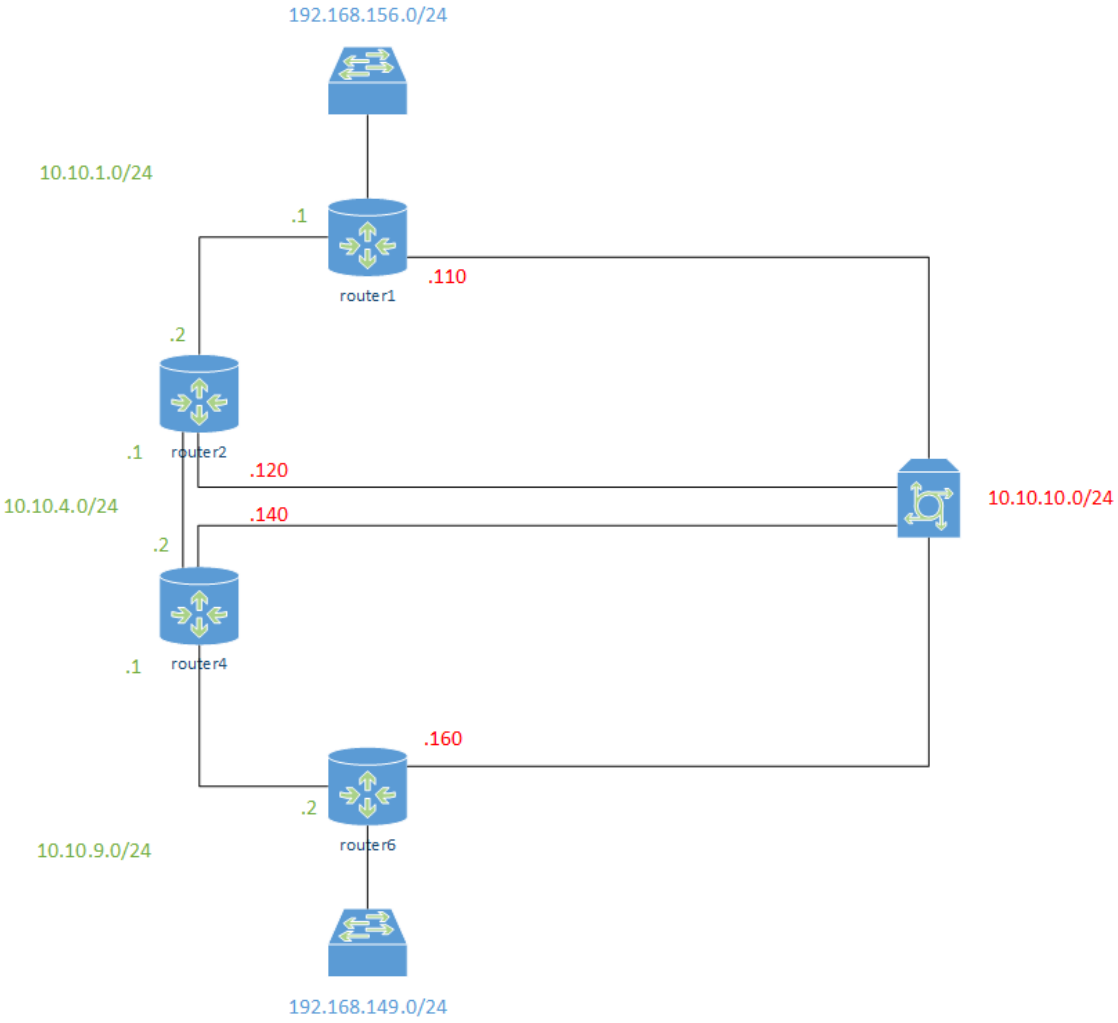
Pro spuštění přes konzoli si opět překopírujeme program do adresáře s SDK knihovnama. Tedy z adresáře /home/cisco/onePK-sdk-1.2.0.173/java/sample-apps/src/main/java/com/cisco /onep/program2 do adresáře /opt/cisco/onep/java/sdk-java-1.2.0.173/java/sample-aps/src/main /java/com/cisco/onep/examples/program2 a provedeme kompilaci.

```
javac com/cisco/onep/examples/program2/program2.java
```

Program spustíme java příkazem.

```
java com.cisco.onep.examples.program2.program2 -a 10.10.10.110 -a2 10.10.10.120
-a3 10.10.10.140 -a4 10.10.10.160 -u cisco1 -p cisco1
```

Pro spuštění zadáváme čtyři směrovače, a proto musíme definovat čtyři IP adresy. Na obrázku 6.1 můžeme vidět celou topologii na které budeme testovat změny QOS v čase.



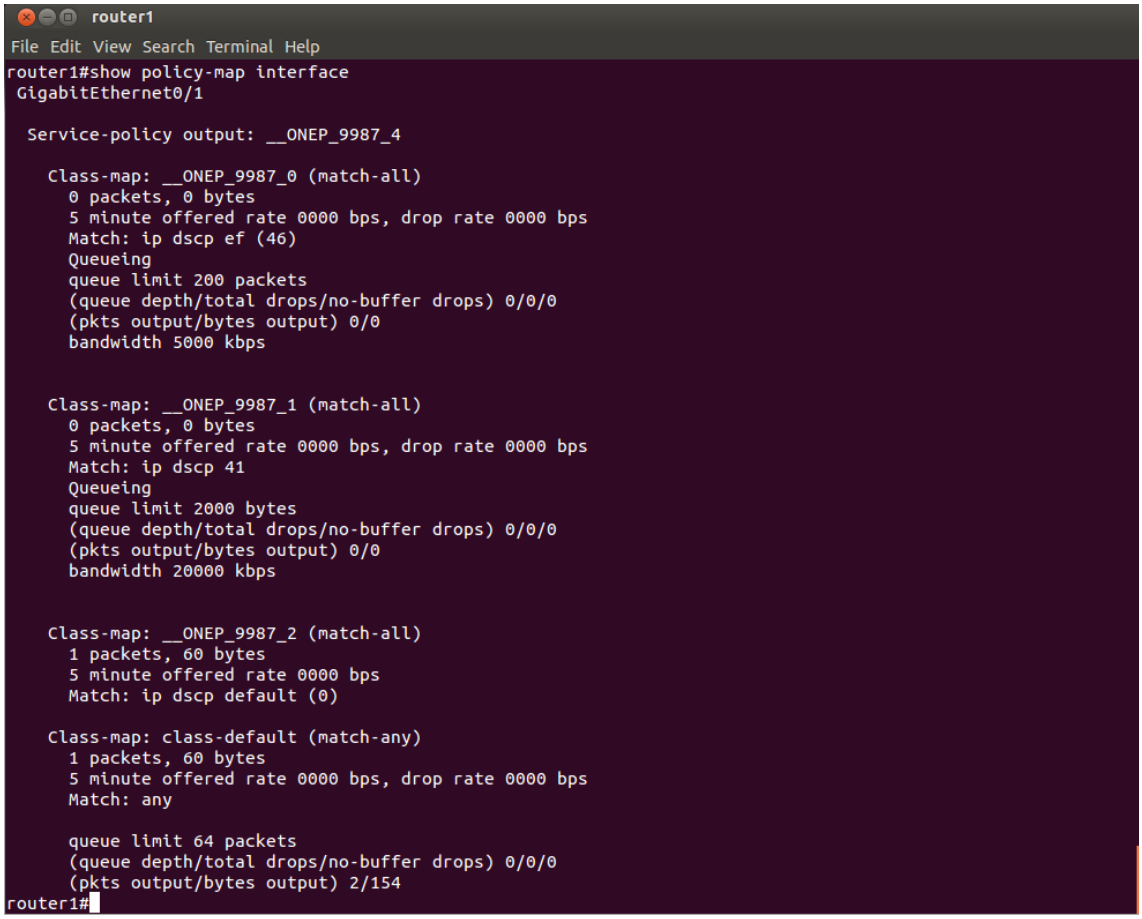
Obrázek 6.1: Logické zapojení topologie při testování QOS v závislosti na čase.

6.3 Chování topologie při použití programu

Při spuštění programu nám program nastaví na všechny směrovače tři třídy. Třídu pro hlas, video a data. Tyto třídy jsou součástí jedné politiky, která tyto třídy spravuje a nastavuje jednotlivé parametry, které tyto třídy obsahují. Data jsou spravována danou třídou podle hodnoty DSCP v záhlaví paketu.

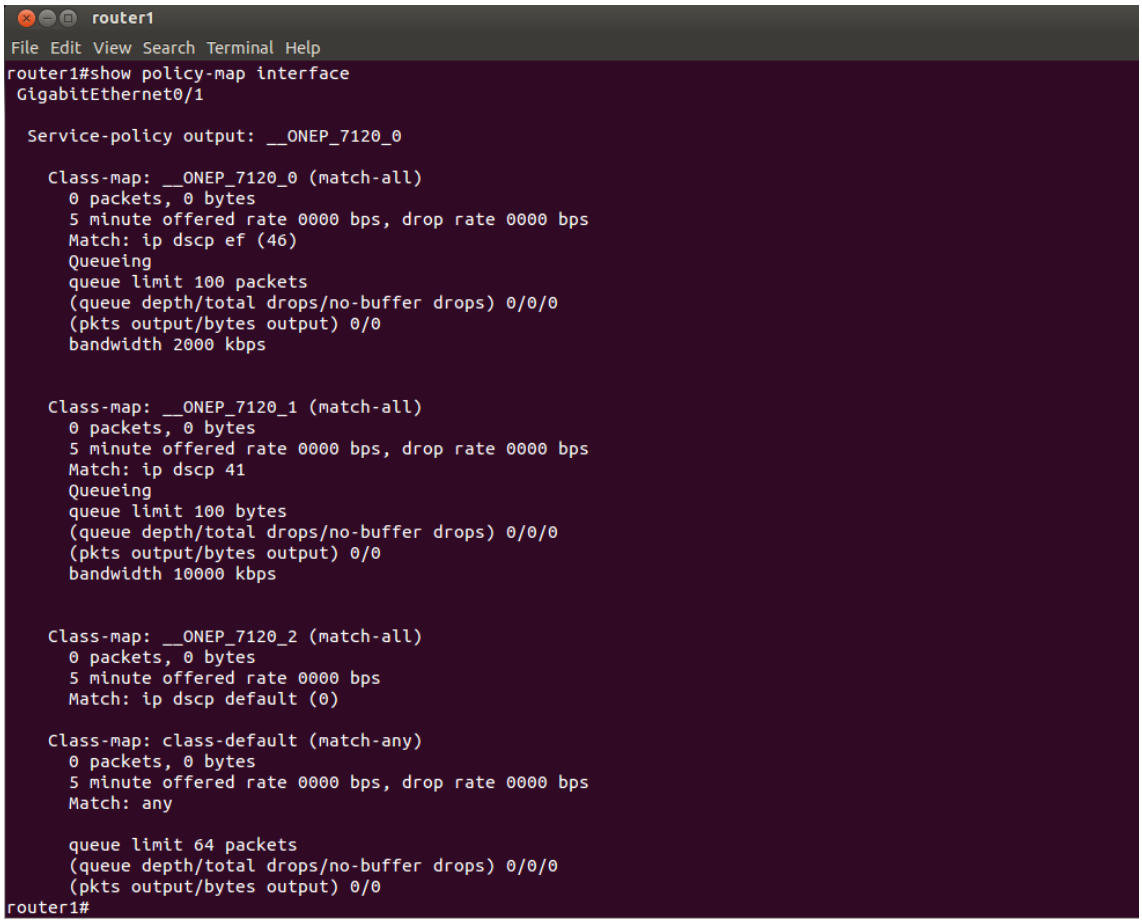
Pokud je hodnota času menší než 16 hodin, pak nám program nastaví pro jednotlivé třídy tyto hodnoty. Pro třídu hlas se nastaví přenosová rychlost 5 000 kbps a velikost fronty na 200 paketů. Tuto třídu poznáme podle nastavené hodnoty DSCP 46, jelikož třídy nejsou na směrovačích aplikovány pod názvem třídy, ale pod číslem procesu pod kterým je program na směrovači spuštěn. Pro třídu video, která je značená hodnotou DSCP 41, program nastaví hodnotu přeno-

sové rychlosti na 20 000 kbps a délku fronty na 2 000 bytů. V rámci nastavení hodnot si můžeme vybrat, zda budeme hodnoty definovat v paketech nebo bytech. Pro třídu data nespecifikujeme žádné hodnoty, takže tato třída bude používat zbytek dostupných zdrojů. Tyto hodnoty můžeme vidět z výstupu směrovače na obrázku 6.2.



Obrázek 6.2: Aplikování QOS profilu 1 na rozhraní.

Když hodnota času překročí hodnotu 16 hodin, tak nám program přenastaví hodnoty v jednotlivých třídách. Pro třídu hlas s hodnotou DSCP 46 se nastaví hodnota přenosové rychlosti z 5 000 kbps na 2 000 kbps a sníží se velikost fronty pro tuto třídu na pouhých 100 paketů. Pro třídu video s hodnotou DSCP 41 se sníží přenosová rychlost z 20 000 kbps na 10 000 kbps a velikost fronty na 100 bytů. Takto dojde k uvolnění prostředků pro třídu data, která používá zbylé dostupné zdroje. Toto přenastavení parametrů v třídách můžeme vidět na obrázku 6.3.



Obrázek 6.3: Aplikování QOS profilu 2 na rozhraní.

Toto chování můžeme využít například pro firemní provoz, kdy během pracovní doby jsou kladeny nároky spíše na VoIP telefonii a videokonference, a naopak po pracovní době dojde k uvolnění prostředků pro data, která bude třeba zálohovat.

7 Přenastavení hodnot traffic shapingu podle vytížení linky

Poslední program, podobně jako druhý, přenastavuje hodnoty QOS pro traffic shaping. V mnoha ohledech funguje program podobně jako program, který nám přenastavoval hodnoty QOS. Také zde dochází k aplikování politiky a jednotlivých tříd, ale hodnoty nejsou ovlivněny časem, ale aktuálním vytížením linky. Hodnoty traffic shapingu jsou nastaveny na základě aktuálního přenosu.

Cílem programu je použití nejefektivnějšího nastavení traffic shapingu pro aktuální situaci. Při traffic shapingu dochází k tvarování provozu tak, že je příchozí provoz ukládán do bufferu, kde jsou pakety setříděny a podle pravidel na základě priorit zasílány. Aby traffic shaping správně fungoval, tak nesmí dojít k přeplnění jednotlivých front, protože pak jsou zahazovány všechny pakety bez ohledu na prioritách. V takovém případě je vhodné přiřadit více zdrojů do prioritních tříd a případně snížit množství provozu, který budeme tvarovat během velikého vytížení linky. Také je vhodné zvětšit délku front pro prioritní provoz.

7.1 Naprogramované funkce

Tento program vznikl vytvořením pomocí funkcí a vlastností z předchozích dvou programů. Z programu pro nastavení hodnot QOS program využívá funkce pro připojení směrovačů a přenastavení politik a tříd. Z programu, který provádí přesměrování na základě hodnoty z rozhraní, pak zjišťuje hodnotu aktuálního vytížení rozhraní, která ovlivňuje chování programu.[9, 10, 11, 12, 13, 14]

1. **Shodné nebo upravené funkce**.

```
public boolean connect(String applicationName)
private SessionConfig createSessionConfig()
public Decision showPinningDialog(String host, String hashType,
String fingerprint, boolean changed)
private boolean readProperties()
private boolean parseCommandLine(String[] args)
public void showAuthenticationDialog()
public List<NetworkInterface> getAllInterfaces()
public void createCapabilities() throws OnepRemoteProcedureException,
OnepConnectionException
public void createmap() throws OnepIllegalArgumentException
public void createpolicy () throws OnepIllegalArgumentException
public void getClassMapResult() throws OnepRemoteProcedureException,
```

```
OnepIllegalArgumentException, OnepConnectionException
public void getPolicyMapResult() throws OnepRemoteProcedureException,
OnepIllegalArgumentException, OnepConnectionException
public void createPolicymapEntry() throws OnepIllegalArgumentException
public NetworkInterface getSpecInterface0()
```

2. **Sledování provozu**

   Funkce pollStatistics byla upravena pro získávání hodnoty vytížení na lince, která nám ovlivňuje, jaké parametry traffic shapingu se nastaví pro jednotlivé třídy. Hodnotu vytíženi, kterou nám Cisco směrovače zobrazují, je v hodnotách paketů za sekundu, bitů za sekundu a nebo procentuálně pomocí hodnot 0 až 255. Tento program nám zjišťuje procentuální vytížení, kde dochází k plynulému měnění hodnot.

   ```
   public void pollStatistics() throws OnepException, InterruptedException
   ```

7.2 Testování programu

Před spuštěním programu pro traffic shaping si opět přerušíme připojení přes pravou větev topologie na obrázku 5.1 tím, že vypneme rozhraní na jednotlivých směrovačích, které mají připojení do této větve na směrovače router3 a router5.

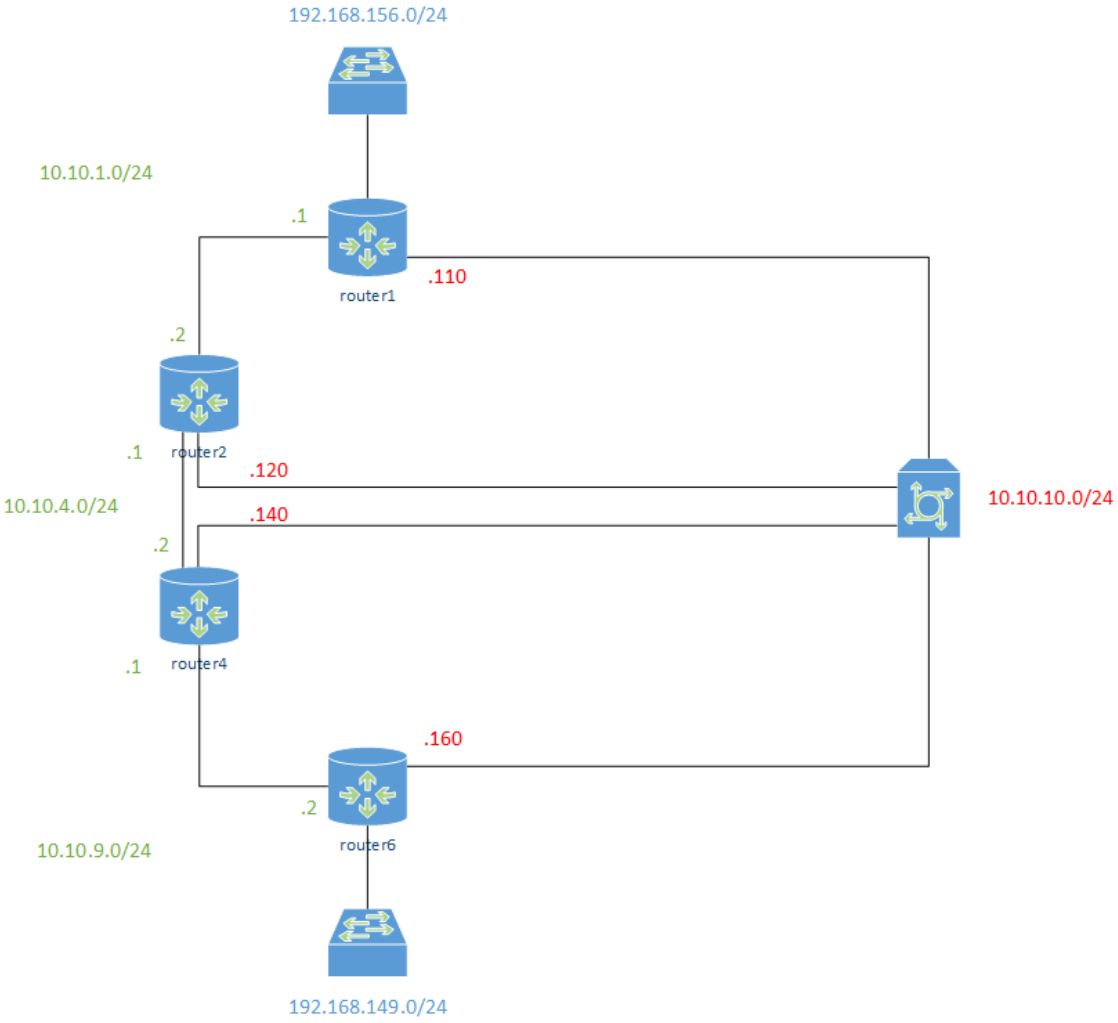
```
enable
conf term
interface Gigabitethernet 0/2
shutdown
end
```

Pro spuštění přes konzoli si opět překopírujeme program do adresáře s SDK knihovnama. Tedy z adresáře /home/cisco/onePK-sdk-1.2.0.173/java/sample-apps/src/main/java/com/cisco /onep/program3 do adresáře /opt/cisco/onep/java/sdk-java-1.2.0.173/java/sample-aps/src/main /java/com/cisco/onep/examples/program3 a provedeme zkompilování programu.

```
javac com/cisco/onep/examples/program3/program3.java
```

Program spustíme pomocí java příkazu.

```
java com.cisco.onep.examples.program3.program3 -a 10.10.10.110 -a2 10.10.10.120
-a3 10.10.10.140 -a4 10.10.10.160 -u cisco1 -p cisco1
```

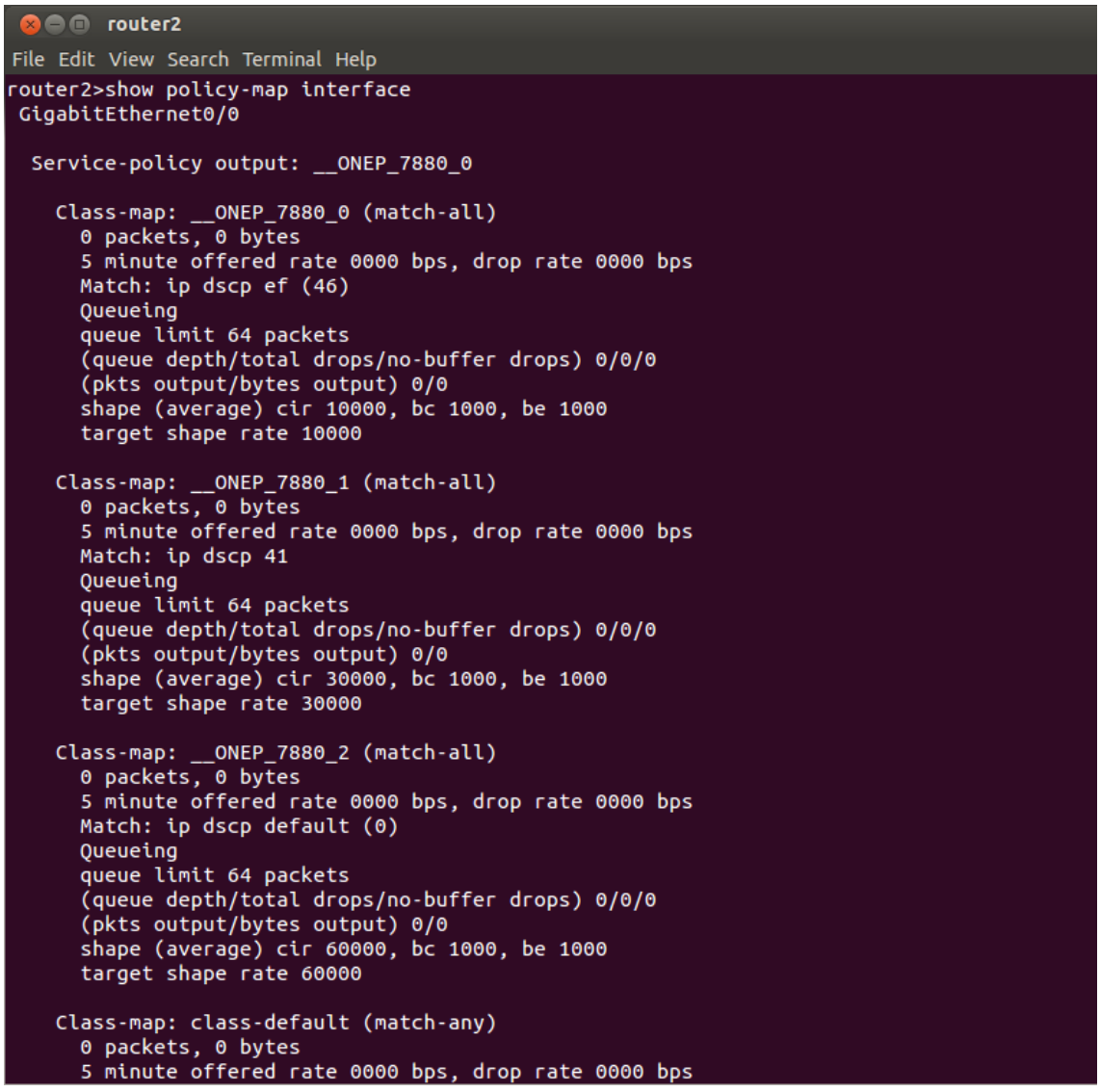


Obrázek 7.1: Logické zapojení topologie při testování programu pro traffic shaping.

Stejně jako u programu pro nastavení QOS parametrů, tak i zde je traffic shaping aplikován na směrovače router1, router2, router4 a router6. Rozdílem je, že algoritmus nenastavuje pouze propustnost a fronty, ale nastavuje hodnotu traffic shappingu. Pokud bude datový tok na lince do hodnoty 500 mbps, pak bude použit jeden profil traffic shappingu. Když bude tato hodnota překročena, dojde k navýšení hodnoty shappingu pro důležitý provoz. V našem případě je to opět provoz hlasu a videa. Topologii, která je použita pro testování, můžeme vidět na obrázku 7.1.

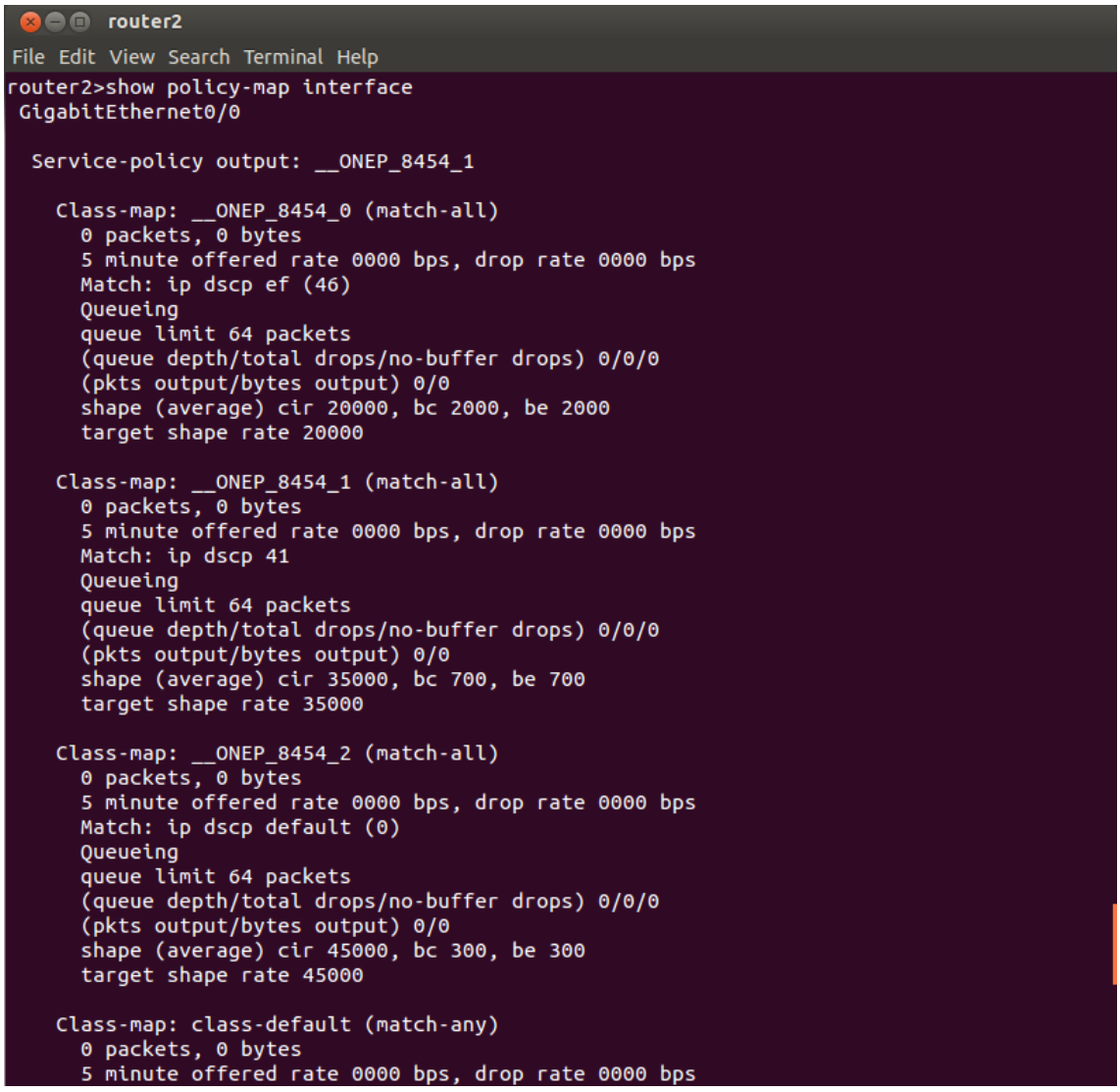
7.3 Chování topologie při použití programu

Když spustíme náš program, tak se nám vytvoří tři třídy. Třída pro hlas s hodnotou DSCP 46, kdy pro tuto třídu nastaví program přenosovou rychlost na 10 000 bps a hodnotu cbs, která nám umožňuje překročit tuto přenosovou rychlost o 1 000 bps. Pro třídu video s DSCP hodnotou 41 se nám rezervuje přenosová rychlost 30 000 bps a hodnotou cbs 1 000 bps. Pro třídu data se rezervuje přenosová rychlost 60 000 bps s hodnotou cbs 1 000 bps. Tyto hodnoty můžeme vyčíst z výstupu směrovače na obrázku 7.2.



Obrázek 7.2: Aplikování profilu 1 pro traffic shaping na první rozhraní.

Pokud nám datový tok překročí hodnotu 500 mbps, tak se nám profil přenastaví. Došlo ke zvýšení hodnoty přenosové rychlosti pro traffic shaping pro třídu hlas z 10 000 na 20 000 bps a hodnota cbs pro překročení přenosové rychlosti se zvýšila na hodnotu 2 000 bps. Pro třídu video se nám zvýší přenosová rychlost pro traffic shaping z 30 000 bps na 35 000 bps, ale hodnota cbs je snížena z 1000 bps na 700 bps. Pro data, která budou použita při tvarování provozu, bude vyhrazena hodnota 45 000 bps a hodnota překročení cbs je o pouhých 300 bps. Tyto hodnoty můžeme vyčíst z obrázku 7.3.



Obrázek 7.3: Aplikování profilu 2 pro traffic shaping na první rozhraní.

Obrázky nám ukazují pouze výpisy z určitého směrovače. Stejné výpisy dostáváme ze všech směrovačů, liší se jen rozhraní na kterém jsou tyto třídy aplikovány.

8 Závěr

Cílem diplomové práce bylo zjistit, jak nové technologie, v tomto případě softwarově definované sítě, mohou pomoci řešit automatizaci v oblasti počítačových sítí a zajistit tak, co možná nejvyšší kvalitu přenosu informací a nejlépe možné efektivní využití dané síťové topologie. I když v rámci diplomové práce se jednalo o použití softwarově definovaných sítí první generace, tak na drobné chyby některých programovatelných API nebo API, které ještě nebyly plně implementovány se ukázalo toto řešení v rámci kvality služeb v sítích velmi efektivní.

V práci se mi podařily vytvořit tři řešení jak využít Cisco one platform kit, tedy Cisco řešení softwarově definovaných sítí pro automatizaci řešení QOS v dané topologii. Kromě QOS v rámci bežného smyslu kdy pojmem QOS myslíme zajištění priority určitého provozu vůči jinému, jsem v práci vytvořil také program, který QOS řešil jako kvalitu provozu v síti z pohledu uživatele. Tedy zajistění co nejvyšší míry spolehlivosti a efektivity dané sítě.

První program se zabývá zajištěním co nejvyšší dostupnosti v síti a její efektivní využití tím, že se snaží odhalit chyby v rámci fyzické vrstvy a provoz přesměrovat skrz zařízení, které neindikují chyby na této první vrstvě OSI modelu. Toto řešení bylo umožněno hlavně díky API rozhraním vytvořeným ke sběru hodnot z jednotlivých směrovačů, které jsou důležité pro bežný troubleshooting v síti správcem. Tedy fyzickou osobou zodpovědnou za správu dané sítě. Díky jiným API rozhraním pak bylo možné přenastavit směrování provozu na bezchybně fungující zařízení. Toto řešení by bylo, bez možnosti použití softwarově definovaných sítí, závislé na nějakém monitoringu v síti a následných změnách v topologii správcem, což je poměrně náročný proces. SDN nám toto řešení umožňují okamžitě během zlomku sekund.

Druhý program se zabýval QOS v rámci běžného pojetí, kdy kvalitou služby myslíme vylepšení parametru jednoho provozu na úkor jiného. Zde jsme si mohli ukázat řešení aplikace QOS na celou topologii nebo její část, kdy tento byl proces plně atomatizovaný. Tím jsme docílili přenastavení parametrů pro přenos v síti ve zlomku sekund a odpadl nám tak složitý proces implementace QOS v dané topologii, kdy bychom museli stanovit přesně stejné parametry na všechna zařízení v síti, aby nám QOS fungovalo správně. Také jsme tímto programem eliminovali proces složitého přenastavování parametů při nevhodném zvolení těchto hodnot pro daný provoz. Samotný program nám ukazuje možnost implementace určitého profilu, kdy uživatelé připojeni do této sítě budou potřebovat upřednostnit určitý druh provozu v danou dobu, ale kdy by tento profil nebyl příliš vhodný mimo tuto dobu kdy je potřebný. Algoritmus tedy ukazuje automatizované přenastavení profilů založené na čase. Toto řešení může být vhodné pro podniky, kdy během pracovní doby je nutné mít upřednostněný jiný provoz než mimo tuto pracovní dobu.

Poslední program nám ukazuje, jak můžeme pomocí one platform kitu ovlivňovat nastavení traffic shapingu na právě probíhajícím provozu. Jedná se o jakési skloubení možností z předchozích dvou programů, kdy z prvního využíváme monitorovaní hodnot na lince a z druhého

přenastavujeme parametry QOS, konkrétněji samotný traffic shaping. Program nám zde monitoruje aktuální vytížení linky a podle vytížení přenastaví hodnoty traffic shapingu. Cílem je ukázat, jak můžeme dosáhnout efektivního využití linky na základě jejího aktuálního stavu. Důvodem pro vznik tohoto programu bylo, že se zvýšující se přenosovou rychlostí a použití traffic shapingu může dojít k vyčerpání zdrojů pro tvarování provozu, a tím k přeplnění front. V takovém případě dojde k zahazování paketů, ať už mají jakoukoli prioritu. Aby k tomuto jevu nedocházelo je nutné upravit fronty. Provoz s nejvyšší prioritou by měl dostat největší počet zdrojů, aby byl zachován, a také můžeme procentuálně snížit množství provozu, který bude tvarován. Při velkém vytížení linky pak bude provoz stále prioritizován bez přeplnění front. Naopak při nízkém vytížení toto nastavení nemusí být příliš efektiviní, a proto můžeme nastavit hodnoty tak, aby byl naopak tvarován všechen provoz. Tento program nám právě ukazuje, jak je možné tohoto chování dosáhnout.

Jak ukazují vytvořené programy, tak softwarově definované sítě jsou ideálním nástrojem, jak automatizovat chování sítě, a zajistit tak její co možná nejefektivnější chování během zlomku sekund a bez nutnosti potřeby vynakládat velké zdroje na přenastavení sítě, která byla již implementována a je použita pro aktivní přenos dat v již produkčním prostředí.

Literatura

[1] *SDxCentral [online].* 23. leden 2015 [cit. 2017-04-20]. Dostupné z: https://www.sdxcentral.com/cisco/datacenter/definitions/cisco-onepk/

[2] *Cisco Blogs [online].* 12. červenec 2013 [cit. 2017-04-20]. Dostupné z: https://blogs.cisco.com/security/ciscos-onepk-part-1-introduction

[3] *Cisco[online].* 20. duben 2017 [cit. 2017-04-20]. Dostupné z: http://www.cisco.com/c/en/us/products/ios-nx-os-software/locator-id-separation-protocol-lisp/index.html

[4] *Cisco Blogs [online].* 20. duben 2017 [cit. 2017-04-20]. Dostupné z: http://www.cisco.com/c/en/us/products/ios-nx-os-software/quality-of-service-qos/index.html

[5] *Samuraj-cz [online].* 20. duben 2017 [cit. 2017-04-20]. Dostupné z: http://www.samuraj-cz.com/clanek/cisco-qos-1-uvod-do-quality-of-service-a-diffserv/

[6] Petr Machník. *Širokopásmové sítě pro integrovanou výuku VUT a VŠB-TUO.*Vyd. 1. Ostrava: Vzsoká škola báňská - Technická univerzita Ostrava, 2014. ISBN 978-80-248-3630-0.

[7] *Wikipedia [online].* 20. duben 2017 [cit. 2017-04-20]. Dostupné z: https://en.wikipedia.org/wiki/Differentiated_services

[8] *Samuraj-cz [online].* 20. duben 2017 [cit. 2017-04-20]. Dostupné z: http://www.samuraj-cz.com/clanek/cisco-qos-4-garance-rychlosti-razeni-do-front-queuing/

[9] Elektronická dokumentace onePK [online]. 20. duben 2017 [cit. 2017-04-20]. Dostupné z: https://developer.cisco.com/media/onePKJavaAPI-v1-1-0/index.html

[10] Elektronická dokumentace onePK [online]. 20. duben 2017 [cit. 2017-04-20]. Dostupné z: https://developer.cisco.com/media/onepk_java_tutorials/basetutorial.html

[11] Elektronická dokumentace onePK [online]. 20. duben 2017 [cit. 2017-04-20]. Dostupné z: https://developer.cisco.com/media/onepk_java_tutorials/SessionTutorial/SessionConfigTutorial.html

[12] Elektronická dokumentace onePK [online]. 20. duben 2017 [cit. 2017-04-20]. Dostupné z: https://developer.cisco.com/media/onepk_java_tutorials/NetworkInterface/interfacestatisticstutorial.html

[13] Elektronická dokumentace onePK [online]. 20. duben 2017 [cit. 2017-04-20]. Dostupné z: https://developer.cisco.com/media/onepk_java_tutorials/Routing/routingsstutorial.html

[14] Elektronická dokumentace onePK [online]. 20. duben 2017 [cit. 2017-04-20]. Dostupné z: https://developer.cisco.com/media/onepk_java_tutorials/Policy/bulkqospolicy.html

9 Obsah CD

Součástí této diplomové práce je CD s elektronickými přílohami. CD obsahuje elektronickou podobu práce v PDF, zdrojové konfigurační soubory použité pro sestavení topologie, java soubory s kódy programů a elektronickou literaturu. Samozřejmostí jsou také obrázky použité v práci a nasnímané v průběhu testování.

Adresářová struktura:

- **lit** - elektronická literatura
  - **doc_html** - dokumentace onePK v html
  - **doc_onepk** - dokumentace onePK v pdf
- **picture** - obrázky z testování a obrázky použité v práci
- **print** - elektronická verze bakalářské práce určená pro tisk
  - **diploma** - zdrojový kód bakalářské práce pro Latex
- **src** - zdrojové kódy programů a skripty
  - **java** - zdrojové java kódy programů
  - **scripts** - zdrojové skripty pro vytvoření topologie

A Elektronické přílohy

A.1 Zdrojové skripty pro sestavení topologie

Zdrojové skripty, které byly použity pro sestavení topologie. Skripty jsou uloženy na přiloženém CD v adresáři *\CD\src\xnode*.

A.2 Zdrojové skripty s návrhem topologie

Zdrojové skripty, které určují jak bude topologie vypadat a jaká bude konfigurace směrovačů při spuštění. Skripty jsou uloženy na přiloženém CD v adresáři *\CD\src\topology*.

A.3 Zdrojové kódy programů

Zdrojové kódy programů jsou napsány v programovacím jazyce Java. Kódy jsou uloženy na přiloženém CD v adresáři *\CD\src\java*. V adresáři java jsou pak tyto kódy rozlišeny podle programů.

A.4 Obrázky pořízené během testování

Obrázky obsahují print screen pořízený během spouštění topologie a během testování kódů jednotlivých programů. Obrázky jsou uloženy na přiloženém CD v adresáři *\CD\picture*.

B Přílohy

B.1 xnode.py

```
#! /usr/bin/python

import commands, os

status, output = commands.getstatusoutput('vmcloud netdiag
xnode')
if "doesn't exist" in output:
    status, output = commands.getstatusoutput('/home/cisco/
    xnode/xnode_certify.sh')
    status, output = commands.getstatusoutput('/home/cisco/
    xnode/xnode_create.sh')
    status, output = commands.getstatusoutput('vmcloud netdiag
    xnode')

title = "Console"
for line in output.split('\n'):
    if line.startswith("Node:"):
title = line.split(".")[2].strip()
    if line.startswith("Console:"):
        os.system('gnome-terminal -t ' + title + ' -e \"' +
        line.split(":")[1].strip() + '\" &')
```

B.2 xnode_certify.sh

```
#!/bin/bash

# TODO: determine the installation strategy for this scipt
bundle and address
# needs to be set to our simpleCA distribution location (or
that has to be in the path)
SIMPLECA_DIR=~/.simpleCA

EXAMPLE_DIR="/home/cisco/vmcloud-example-networks/xnode"
```

```
#EXAMPLE_DIR="/tftpboot"

$SIMPLECA_DIR/createNEp12.sh -un router1 -dns router1.xnode.
example.com -ip 10.10.10.110 -pass cisco -out $EXAMPLE_DIR/
router1.p12
if [ $? -ne 0 ] ; then echo "FAIL: perhaps simpleCA is not
configured?" ; exit ; fi
$SIMPLECA_DIR/createNEp12.sh -un router2 -dns router2.xnode.
example.com -ip 10.10.10.120 -pass cisco -out $EXAMPLE_DIR/
router2.p12
if [ $? -ne 0 ] ; then echo "FAIL: perhaps simpleCA is not
configured?" ; exit ; fi
$SIMPLECA_DIR/createNEp12.sh -un router3 -dns router3.xnode.
example.com -ip 10.10.10.130 -pass cisco -out $EXAMPLE_DIR/
router3.p12
if [ $? -ne 0 ] ; then echo "FAIL: perhaps simpleCA is not
configured?" ; exit ; fi
$SIMPLECA_DIR/createNEp12.sh -un router4 -dns router4.xnode.
example.com -ip 10.10.10.140 -pass cisco -out $EXAMPLE_DIR/
router4.p12
if [ $? -ne 0 ] ; then echo "FAIL: perhaps simpleCA is not
configured?" ; exit ; fi
$SIMPLECA_DIR/createNEp12.sh -un router5 -dns router5.xnode.
example.com -ip 10.10.10.150 -pass cisco -out $EXAMPLE_DIR/
router5.p12
if [ $? -ne 0 ] ; then echo "FAIL: perhaps simpleCA is not
configured?" ; exit ; fi
$SIMPLECA_DIR/createNEp12.sh -un router6 -dns router6.xnode.
example.com -ip 10.10.10.160 -pass cisco -out $EXAMPLE_DIR/
router6.p12
if [ $? -ne 0 ] ; then echo "FAIL: perhaps simpleCA is not
configured?" ; exit ; fi

echo "Generated xnode router certificates as pkcs12 files"
```

B.3 xnode_create.sh

```
#!/bin/bash
VIRL_FILE="/home/cisco/vmcloud-example-networks/xnode/xnode.
virl"
VMNET_NAME="xnode"
vmcloud netcreate -v $VIRL_FILE $VMNET_NAME |
zenity --progress --auto-kill --pulsate --no-cancel --auto-
close \
--text "Running:  vmcloud netcreate -v $VIRL_FILE $VMNET_NAME
" \
--title 'Creating a xnode virtual network'

 MESSAGE=$(cat /var/vmcloud/.vmCloud_cisco/xnodevmlog.log)
 cat /var/vmcloud/.vmCloud_cisco/xnodevmlog.log | grep Success
if [[ $? -eq 0 ]]; then
    zenity --info \
        --text='The "xnode" network is created' --title "xnode
network"
else
    zenity --error \
        --text="`printf "Failed to create the xnode network \n
\n $MESSAGE"`"
fi
```

B.4 xnode_delete

```
#!/bin/bash
#
# Deletes the xnode topology

vmnet_name='xnode'
#sudo service tftpd-hpa stop
vmcloud netdelete "${vmnet_name}" \
    | zenity --progress --auto-kill --pulsate --no-cancel --
auto-close \
        --text="Running:  vmcloud netdelete ${vmnet_name}" \
        --title='Deleting the "xnode" virtual network'
```

```
if [[ $? -eq 0 ]]; then
    zenity --info \
        --text='The "xnode" network was deleted'
else
    zenity --error \
        --text='Failed to delete the "xnode" network'
fi
```

B.5 xnode_info.sh

```
#!/bin/bash

vmcloud netdiag xnode | zenity --text-info --title "Network
info" --width=450 --height=400
```

B.6 netlist.sh

```
#!/bin/bash

vmcloud netlist | zenity --text-info --title "Network
topologies" --width=450 --height=400
```

B.7 xnode.virl

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>\newline
<topology xmlns="http://www.cisco.com/VIRL" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" schemaVersion="0.3" xsi:schemaLocation="http://www
.cisco.com/VIRL http://cide.cisco.com/vmmaestro/schema/virl.xsd">\newline
    <node name="router1" type="SIMPLE" subtype="vios" location="100,100" vmImage="
/usr/share/vmcloud/data/images/vios.ova">
        <extensions>
            <entry key="bootstrap configuration" type="String">/home/cisco/vmcloud-
example-networks/xnode/router1.con</entry>
            <entry key="import files" type="String">/home/cisco/vmcloud-example-
networks/xnode/router1.p12</entry>
        </extensions>
        <interface name="GigabitEthernet0/0"/>
```

```
        <interface name="GigabitEthernet0/1"/>
        <interface name="GigabitEthernet0/2"/>
<interface name="GigabitEthernet0/3"/>
    </node>
    <node name="router2" type="SIMPLE" subtype="vios" location="100,200" vmImage="
/usr/share/vmcloud/data/images/vios.ova">
        <extensions>
            <entry key="bootstrap configuration" type="String">/home/cisco/vmcloud
-example-networks/xnode/router2.con</entry>
            <entry key="import files" type="String">/home/cisco/vmcloud-example-
networks/xnode/router2.p12</entry>\newline
        </extensions>
        <interface name="GigabitEthernet0/0"/>
        <interface name="GigabitEthernet0/1"/>
        <interface name="GigabitEthernet0/2"/>
        <interface name="GigabitEthernet0/3"/>
        <interface name="GigabitEthernet0/4"/>
    </node>\newline
    <node name="router3" type="SIMPLE" subtype="vios" location="100,300" vmImage="
/usr/share/vmcloud/data/images/vios.ova">
        <extensions>
            <entry key="bootstrap configuration" type="String">/home/cisco/vmcloud-
example-networks/xnode/router3.con</entry>\newline
            <entry key="import files" type="String">/home/cisco/vmcloud-example-
networks/xnode/router3.p12</entry>
        </extensions>
        <interface name="GigabitEthernet0/0"/>
        <interface name="GigabitEthernet0/1"/>
        <interface name="GigabitEthernet0/2"/>
        <interface name="GigabitEthernet0/3"/>
        <interface name="GigabitEthernet0/4"/>
    </node>
<node name="router4" type="SIMPLE" subtype="vios" location="100,400" vmImage="
/usr/share/vmcloud/data/images/vios.ova">
        <extensions>
            <entry key="bootstrap configuration" type="String">/home/cisco/vmcloud-
example-networks/xnode/router4.con</entry>
```

```
            <entry key="import files" type="String">/home/cisco/vmcloud-example-
networks/\newline xnode/router4.p12</entry>
        </extensions>
        <interface name="GigabitEthernet0/0"/>
        <interface name="GigabitEthernet0/1"/>
        <interface name="GigabitEthernet0/2"/>
        <interface name="GigabitEthernet0/3"/>
        <interface name="GigabitEthernet0/4"/>
    </node>
<node name="router5" type="SIMPLE" subtype="vios" location="100,500" vmImage="
/usr/share/vmcloud/data/images/vios.ova">
        <extensions>
            <entry key="bootstrap configuration" type="String">/home/cisco/vmcloud-
example-networks/xnode/router5.con</entry>
            <entry key="import files" type="String">/home/cisco/vmcloud-example-
networks/xnode/router5.p12</entry>\newline
        </extensions>
        <interface name="GigabitEthernet0/0"/>
        <interface name="GigabitEthernet0/1"/>
        <interface name="GigabitEthernet0/2"/>
        <interface name="GigabitEthernet0/3"/>
        <interface name="GigabitEthernet0/4"/>
    </node>
<node name="router6" type="SIMPLE" subtype="vios" location="100,600" vmImage="
/usr/share/vmcloud/data/images/vios.ova">
        <extensions>
            <entry key="bootstrap configuration" type="String">/home/cisco/vmcloud-
example-networks/xnode/router6.con</entry>
            <entry key="import files" type="String">/home/cisco/vmcloud-example-
networks/xnode/router6.p12</entry>
        </extensions>
        <interface name="GigabitEthernet0/0"/>
        <interface name="GigabitEthernet0/1"/>
        <interface name="GigabitEthernet0/2"/>
        <interface name="GigabitEthernet0/3"/>
    </node>
    <node name="lan\_ex\_2" type="SEGMENT" location="374,520"/>
```

```
    <node name="lan\_ex" type="SEGMENT" location="722,161"/>
    <node name="eth1" type="ASSET" location="671,235">
        <interface name="none0"/>
        <interface name="none1"/>
    </node>
    <node name="eth0" type="ASSET" location="600,200">
        <interface name="none0"/>
        <interface name="none1"/>
    </node>
    <node name="vmc\_lan\_1" type="SEGMENT" location="374,620"/>
    <connection src="/topology/node[1]/interface[1]" dst="/topology/node[10]/
inter-face[1]"/>
    <connection src="/topology/node[1]/interface[2]" dst="/topology/node[2]/
inter-face[2]"/>
    <connection src="/topology/node[1]/interface[3]" dst="/topology/node[3]/
inter-face[3]"/>
    <connection src="/topology/node[1]/interface[4]" dst="/topology/node[11]"/>
    <connection src="/topology/node[2]/interface[1]" dst="/topology/node[4]/
inter-face[1]"/>
    <connection src="/topology/node[2]/interface[3]" dst="/topology/node[5]/
inter-face[3]"/>
    <connection src="/topology/node[2]/interface[4]" dst="/topology/node[3]/
inter-face[4]"/>
    <connection src="/topology/node[2]/interface[5]" dst="/topology/node[11]"/>
    <connection src="/topology/node[3]/interface[1]" dst="/topology/node[5]/
inter-face[1]"/>
    <connection src="/topology/node[3]/interface[2]" dst="/topology/node[4]/
inter-face[2]"/>
    <connection src="/topology/node[3]/interface[5]" dst="/topology/node[11]"/>
    <connection src="/topology/node[4]/interface[3]" dst="/topology/node[6]/
inter-face[3]"/>
    <connection src="/topology/node[4]/interface[4]" dst="/topology/node[5]/
inter-face[4]"/>
    <connection src="/topology/node[4]/interface[5]" dst="/topology/node[11]"/>
    <connection src="/topology/node[5]/interface[2]" dst="/topology/node[6]/
inter-face[2]"/>
    <connection src="/topology/node[5]/interface[5]" dst="/topology/node[11]"/>
```

```
    <connection src="/topology/node[6]/interface[1]" dst="/topology/node[9]/
inter-face[1]"/>
    <connection src="/topology/node[6]/interface[4]" dst="/topology/node[11]"/>
    <connection src="/topology/node[8]" dst="/topology/node[9]/interface[2]"/>
    <connection src="/topology/node[7]" dst="/topology/node[10]/interface[2]"/>
</topology>
```

B.8 Konfigurace

Příklad startup skriptu pro směrovač

```
version 15.3\newline
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname router1
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
mmi polling-interval 60
no mmi auto-configure
no mmi pvc
mmi snmp-timeout 180
!
!
!
!
!
!
ip domain lookup source-interface GigabitEthernet0/0
ip name-server 192.168.156.2
ip cef
```

```
no ipv6 cef
ipv6 multicast rpf use-bgp
!\newline
multilink bundle-name authenticated
!
!
!
username cisco1 privilege 15 password 0 cisco1
!
redundancy
!
!
!
!
!
!
!
!
!
!
!
!
!
!
!
interface GigabitEthernet0/0
 ip address dhcp
 no shutdown
 duplex auto
 speed auto
!
interface GigabitEthernet0/1
 ip address 10.10.1.1 255.255.255.0
 no shutdown
 duplex auto
 speed auto
!
interface GigabitEthernet0/2
```

```
 ip address 10.10.2.1 255.255.255.0
 no shutdown
 duplex auto
 speed auto
!
interface GigabitEthernet0/3
 ip address 10.10.10.110 255.255.255.0
 no shutdown
 duplex auto
 speed auto
!\newline
router ospf 1
 network 10.10.1.0 0.0.0.255 area 0
 network 10.10.2.0 0.0.0.255 area 0
 network 192.168.0.0 0.0.0.255 area 0
 network 192.168.156.0 0.0.0.255 area 0
 default-information originate
!
ip forward-protocol nd
!
!
no ip http server
no ip http secure-server
!
!
!
!
control-plane
!
banner exec \^C
**************************************************************************
* vIOS - Cisco Systems Confidential                                      *
* Unauthorized use or distribution of this software is expressly         *
* Prohibited.                                                            *
**************************************************************************
banner incoming \^C
**************************************************************************
```

```
* vIOS - Cisco Systems Confidential                                    *
* Unauthorized use or distribution of this software is expressly       *
* Prohibited.                                                          *
************************************************************************\^C
banner login \^C
************************************************************************
* vIOS - Cisco Systems Confidential                                    *
* Unauthorized use or distribution of this software is expressly       *
* Prohibited.                                                          *
************************************************************************\^C
!
line con 0
line aux 0
line vty 0 4
 transport input all
!
onep
 transport type tls localcert demoTP disable-remotecert-
validation
 start
!
!
! IOS PKI will fail to import the tftp file if we attempt this
before
! the config has been fully applied. So if we just do:
!   crypto pki import demoTP pkcs12 [location] [etc...]
! We would see something similar to this in the boot log:
!   *Nov 29 19:27:32.415: CRYPTO\_PKI: Copying pkcs12 from
flash1://bootstrap\_admin.con
!   *Nov 29 19:27:32.492: %PKI-6-PKCS12IMPORT\_FAIL: PKCS #12
Import Failed.
! Therefore we use a short delay before loading the pkcs12
file:
!
event manager applet load\_identity
 event timer countdown name Delay time 20
 action 0.0 cli command "enable"
```

```
 action 1.0 cli command "config terminal"
 action 2.0 cli command "file prompt quiet"
 action 3.0 cli command "crypto pki import demoTP pkcs12
flash2://router1.p12 password cisco"
 action 4.0 syslog msg "Loaded bootstrap identity certificate"
!
end
```