

Slovenská technická univerzita

Fakulta informatiky a informačných technológií
Ilkovičova 2, 812 19 Bratislava

Dokumentácia k inžinierskemu dielu

Tímový projekt I

Sealfisticated Networkers

Číslo a názov tímu: 21. – Sealfisticated Networkers
Vedúci tímu: Ing. Peter Trúchly, PhD.
Členovia tímu: Bc. Maroš Hrobák, Bc. Matúš Kováč, Bc. Hana Kuntová, Bc. Marko Ondruš, Bc. Erika Štefanková, Bc. Matej Uhlík, Bc. Peter Válka
Akademický rok: 2017/2018

Obsah

Slovník.....	V
Zoznam skratiek	VI
1. Úvod	1
1.1. Ciele pre zimný semester.....	1
2. Softvérovo definované siete	2
2.1. Technológia SDN	2
2.1.1. Vlastnosti SDN.....	2
2.2. Protokol OpenFlow	4
2.3. SDN forwardery	7
2.3.1. Prepínač	7
2.3.2. Open vSwitch (OVS).....	9
2.3.3. Pantou/OpenWRT	9
2.3.4. OFSoftSwitch13	9
2.3.5. Indigo Virtual Switch (IVS)	10
2.3.6. Linc.....	10
2.4. SDN kontroléry	10
2.4.1. NOX	10
2.4.2. POX	11
2.4.3. Beacon	12
2.4.4. Floodlight	13
2.4.5. MUL	13
2.4.6. Maestro	14
2.4.7. Iné	14
2.5. Kontrolér Ryu.....	15
2.6. Simulačné/emulačné prostredia.....	16
2.6.1. EstiNet	16
2.6.2. Ns-3	17
2.6.3. Trema.....	17
2.6.4. Mininet	17
3. Generátory premávky	19
3.1. Hping	19
3.2. Ostinato.....	20
3.3. Scapy	21
3.4. packETH.....	22
4. QoS z hľadiska poskytovania služieb na sieti	24
4.1. VOIP.....	24
4.1.1. Latencia	24
4.1.2. Odchýlka.....	24
4.1.3. Stratovosť paketov.....	24
4.2. Video interaktivita	25
4.2.1. Latencia	25
4.2.2. Stratovosť paketov.....	25
4.2.3. Odchýlka.....	25
4.2.4. Šírka pásma	25
4.3. Video stream.....	26
4.3.1. Latencia	26
4.3.2. Stratovosť paketov.....	26

4.3.3.	Odchýlka.....	26
4.3.4.	Šírka pásma	26
4.4.	Dáta.....	27
5.	Testovanie.....	28
5.1.	Testovací scenár č.1 (Hlavný testovací scenár).....	28
5.1.1.	Testovacie prostredie.....	28
5.1.2.	Testovacia topológia.....	29
5.1.3.	Zhodnotenie.....	29
5.2.	Testovací scenár č.2 (QoS).....	31
5.2.1.	Testovacie prostredie.....	31
5.2.2.	Testovacia topológia.....	31
5.2.3.	Zhodnotenie.....	31
5.3.	Testovací scenár č.3 (Mininet - wifi).....	32
5.3.1.	Testovacie prostredie.....	32
5.3.2.	Testovacia topológia.....	32
5.3.3.	Zhodnotenie.....	33
5.4.	Testovací scenár č.4.....	33
5.4.1.	Testovacie prostredie.....	33
5.4.2.	Testovacia topológia.....	34
5.4.3.	Zhodnotenie.....	34
6.	Literatúra	36
7.	Príloha A – Inštalčná príručka	38
A.1	Inštalácia Ryu	38
8.	Príloha B – Používateľská príručka	39
B.1	Mininet.....	39
9.	Príloha C – Priebeh testovania.....	42
C.1	Testovací scenár č.1	42
C.2	Testovací scenár č.2.....	43
C.3	Testovací scenár č.3.....	44
C.4	Testovací scenár č.4.....	44

Slovník

Slovo	Vysvetlenie
Cookie	Malé množstvo dát, ktoré WWW server pošle prehliadaču, ktorý ho uloží na počítači
Debug	Postup pre nájdenie a zmenšovanie chýb
Forwarder	Prepínač v softvérovo definovaných sieťach
Framework	Aplikačný rámec
Open source	Voľne šíriteľný zdrojový kód
Plug in	Prídavný modul
Root bridge	Prepínač, ktorý je koreň stromu
Video stream	Video prenos
Wildcard	Rozmedzie hodnôt

Zoznam skratiek

Skratka	Vysvetlenie
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IT	Internetové technológie
OVSDB	Databáza Open vSwitch
QoS	Kvalita služieb
SDN	Softvérovo definované siete
SLA	Service Level Agreement
SSH	Zabezpečený prístup k príkazovému interpretovaču
SSL	Vrstva bezpečných paketov
TCP	Protokol riadenia prenosu
UDP	User datagram protocol
VRRP	Virtual Router Redundancy Protocol

1. Úvod

Súčasný rýchly rozvoj internetu, techniky a počítačových systémov priniesol okrem iného aj množstvo zariadení na prevádzku a správu počítačových a komunikačných sietí, ako napríklad prepínače a smerovače. Každým dňom sa zvyšuje počet ľudí pripojených do internetu a preto je veľké dátové centrá v IT spoločnostiach stále náročnejšie manažovať zväčšujúci sa nárast prenosovej a výpočtovej kapacity. S týmto problémom rastie aj potreba jednoduchej škálovateľnosti zariadení. Mnoho výrobcov sieťového hardvéru prichádza stále s novými inováciami ako tento problém doriešiť, avšak ani jedno z nich neponúka komplexné vyriešenia náročnej úlohy.

Jedným z riešení sú softvérovo definované siete, ktoré umožňujú riadiť, meniť a manažovať správanie siete dynamicky cez rozhranie. Hlavným bodom SDN architektúr je centralizovaný bod riadenia a manažmentu komunikačnej a počítačovej siete nazývaný SDN kontrolér. Tento kontrolér sa nachádza na vrchu celej siete a pomocou neho je možné jednoducho, inteligentne a efektívne riadiť smerovanie a využívanie sieťových zdrojov. Motiváciou pre tento projekt je zabezpečiť implementáciu algoritmov ktoré dokážu nielen nájsť vhodnú cestu sieťou pre každý tok, ale aj zohľadniť kvalitatívne parametre tokov alebo realizovať optimálne rozdeľovanie zátáže v sieti.

Tento dokument ponúka okrem „big picture“ projektu aj ciele vytýčené v zimnom semestri, podrobnú analýzu problematiky softvérovo definovaných sietí, návrh testovacích scenárov spolu s ich vyhodnotením, vzhľadom na následnú implementáciu vybraných algoritmov, ktoré budú efektívne riadiť zátáž v sieti.

1.1. Ciele pre zimný semester

1. Oboznámiť s problematikou softvérovo definovaných sietí
2. Nainštalovať celé potrebné prostredie
3. Overiť náležitosti prostredia potrebné pre implementáciu
4. Získať zručnosti a základné princípy fungovania softvérovo definovaných sietí
5. Implementovať jednu metódu

2. Softvérovo definované siete

Kapitola opisuje princíp fungovania softvérovo definovaných sietí. Vysvetľuje základné vlastnosti, ako aj protokol OpenFlow, či popisuje a porovnáva forwarderi, či prepínače alebo kontroléri.

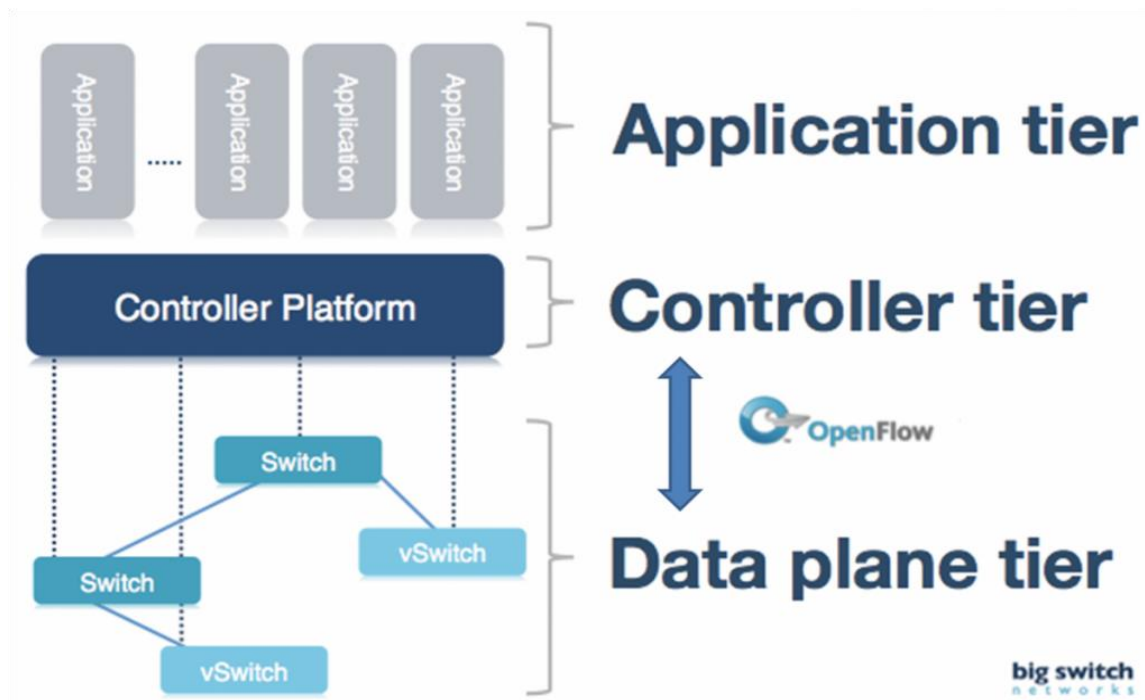
2.1. Technológia SDN

Inicializovať, kontrolovať, meniť a manažovať správanie siete dynamicky cez rozhranie.

2.1.1. Vlastnosti SDN

Vlastnosti SDN sietí sú nasledovné [30] [31] [32]:

- **Priamo programovateľné** – sieťová kontrola je priamo programovateľná pretože je oddelená od smerovacích funkcií
- **Agilná** – oddelenie kontroly od smerovania umožňuje administrátorom dynamicky prispôbovať sieťovú premávku aby splňala meniace sa potreby
- **Centrálne manažovateľná** – sieťová inteligencia je centralizovaná v SDN (Software-Defined Networking) kontroléroch, ktoré udržujú celkový pohľad na sieť. Toto sa ukazuje vo viacerých aplikáciách ako jeden logický prepínač.
- **Programovateľná** – umožňuje meniť konfiguráciu, manažovať, zabezpečiť alebo optimalizovať sieť rýchlo cez automatické SDN programy, ktoré si môžu ľudia písať sami, pretože nie sú proprietárne.



Obrázok 1. SDN architektúra [31]

Dátová rovina

Je zodpovedná za narábanie s paketmi na základe príkazov z riadiacej roviny. Môže zahadzovať, meniť a preposielať pakety. [30] [31] [32]

Operačná rovina

Je zodpovedná za informácie ako, či je daný forwarder zapnutý alebo vypnutý, koľko voľných portov má, stav na každom porte a iné. Táto rovina je koncovým bodom manažovacej roviny. Táto rovina referuje ku zdrojom sieťových zariadení ako porty, pamäť, atď. Môže to byť kľúčne aj súčasť smerovacej roviny, ale definícia roviny je o rozdelení operácií nad nejakou časťou. [30] [31] [32]

Riadiaca rovina

Je zodpovedná za rozhodnutia, kam by mal byť paket poslaný jedným alebo viacerými sieťovými zariadeniami a tlačí tieto rozhodnutia dole na forwardery na vykonanie. Táto rovina sa hlavne zameriava na smerovaciu rovinu, a informácie pre ňu. Môže ale používať informácie z operačnej roviny, pre rozhodnutia o tom ako je daný port vyťažený, atď. Jeho hlavnou úlohou je vyladiť smerovaciu tabuľku ktorá ovláda smerovaciu rovinu, na základe sieťovej topológie alebo externých servisných žiadostí. Medzi riadiacou rovinou, čo je niekedy nazývaná aj ako „east-west“ interface, sa používa napr. BGP. [30] [31] [32]

Úlohou riadiacej roviny je [30] [31] [32]:

- Zistenie topológie a jej údržba
- Výber trasy paketu a konkretizácia
- Mechanizmus prerušenia trasy

Manažovacia rovina

Je zodpovedná za monitorovanie, konfigurovanie a udržiavanie sieťových zariadení. Zameriava sa hlavne na riadiacu rovinu. Môže byť použitá na konfiguráciu smerovacej roviny ale robí to málo krát, je komplikovanejšia ako riadiaca rovina. Napríklad môže nastaviť časť alebo celú tabuľku smerovacích pravidiel, ale táto akcia bude braná ako doporučenie pre riadiacu rovinu, a nie ako prikazovanie. Hlavnou úlohou manažovacej roviny je [30] [31] [32]:

- Správa chýb a ich monitorovanie
- Správa konfigurácie

Aplikačná rovina

Miesto kde sa aplikácie a servisy ktoré definujú sieťové správanie nachádzajú. Aplikácie, ktoré priamo podporujú operáciu smerovacej roviny (napríklad smerovanie v riadiacej rovine) nie sú chápané ako časť aplikačnej vrstvy. Aplikácie môžu byť implementované tak, aby ovplyvňovali viacero vrstiev. [30] [31] [32]

2.2. Protokol OpenFlow

Pre praktické uplatnenie SDN sietí je potrebné splniť 2 požiadavky[12] [13]:

- v sieti musí byť spoločná logická architektúra v rámci všetkých prepínačov, smerovačov a iných sieťových zariadeniach, riadených SDN kontrolérom
- je potrebný bezpečný protokol medzi SDN kontrolérom a sieťovými zariadeniami

Obe tieto podmienky rieši OpenFlow. Stručne sa teda dá povedať, že je to protokol medzi kontrolérom a sieťovými zariadeniami, ako aj nástroj pre špecifikáciu logickej štruktúry siete. Ako už vyplýva z vyššie spomenutého, kontrolér a prepínač sú dve rozdielne súčasti týchto sietí. Kontrolér slúži na spravovanie siete „na diaľku“, pričom ovláda práve prepínače, ktorým posiela príkazy a smeruje do nich pakety.[12] [13]

Pracuje v rámci TCP (Transmission Control Protocol), kde by mal počúvať na porte 6653, na ktorý sa hlásia prepínače, ktoré chcú nadviazať spojenie. Vzdialené riadenie prebieha na vrstve 3, kde sa môžu vykonať rôzne akcie[12] [13]:

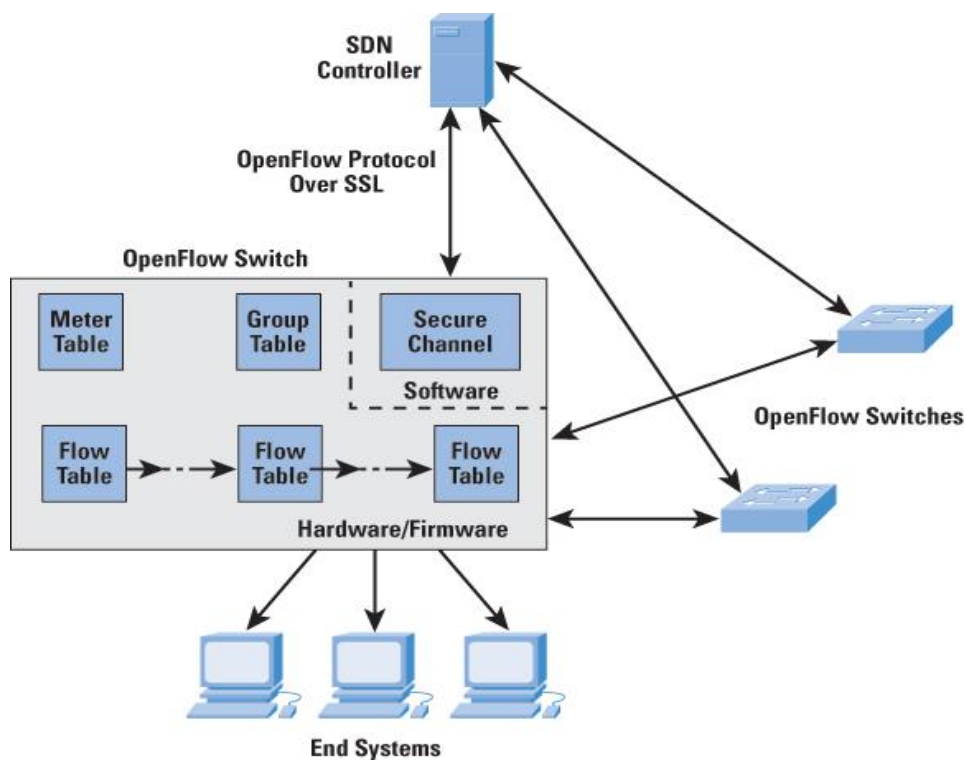
- pridávanie, zmena alebo vyradovanie paketov, podľa vopred definovaných pravidiel a akcií
- smerovanie akceptovaných paketov prepínačom
- neakceptované pakety sú smerované do kontroléra, ktorý môže:
 - zmeniť pravidlá smerovacej tabuľky na jednom alebo viacerých prepínačoch
 - nastaviť nové pravidlá, aby tak predišiel veľkej komunikácii medzi prepínačom a kontrolérom

V SDN sieťach sa vyskytuje viacero typov tabuliek, ako napríklad[12] [13]:

- prietoková tabuľka – spája pakety do určitých tokov a špecifikuje funkcie, ktoré sa majú vykonať na paketoch (môže ich byť aj viac)
- skupinová tabuľka – prietoková tabuľka sem môže smerovať tok, čo spustí viacero akcií, ktoré ovplyvňujú jeden alebo viacero tokov
- metrová tabuľka – môže spustiť viacero rôznych činností súvisiacich s výkonom na toku

Každá z tabuliek má určité súčasti, na základe ktorých pracujú. Ako príklad si môžeme uviesť súčasti prietokovej tabuľky[12] [13]:

- vyhovujúce polia – na ich základe vyberá vyhovujúce pakety
- priorita – relatívna priorita záznamov v tabuľke
- počítadlá – rôzne počítadlá a časovače definované OpenFlow
- inštrukcie – akcia, ktorá sa má vykonať, ak nastane zhoda
- pauzy – maximálny čas nečinnosti prepínača
- cookie – potrebné pre filtre, zmenu toku alebo jeho zmazanie (nepoužívajú sa, keď sa pakety spracovávajú)
- tabuľka chýbajúcich tokov – wildcard pre vyhovujúce polia



Obrázok 2. OpenFlow komunikácia [11]

Obrázok popisuje približnú komunikáciu v rámci OpenFlow protokolu[12] [13]:

- SDN kontrolér komunikuje s prepínačmi, ktoré sú kompatibilné s OpenFlow, pomocou OpenFlow protokolu bežiaceho cez SSL (Secure Sockets Layer)
- každý prepínač sa pripojí k zariadeniam cieľového používateľa, ktoré sú zdrojmi a cieľmi paketových tokov
- každý prepínač má niekoľko tabuliek (spomenutých vyššie), implementovaných hardvérom alebo firmvérom, ktoré sa používajú na riadenie toku cez prepínače

Bezpečnosť

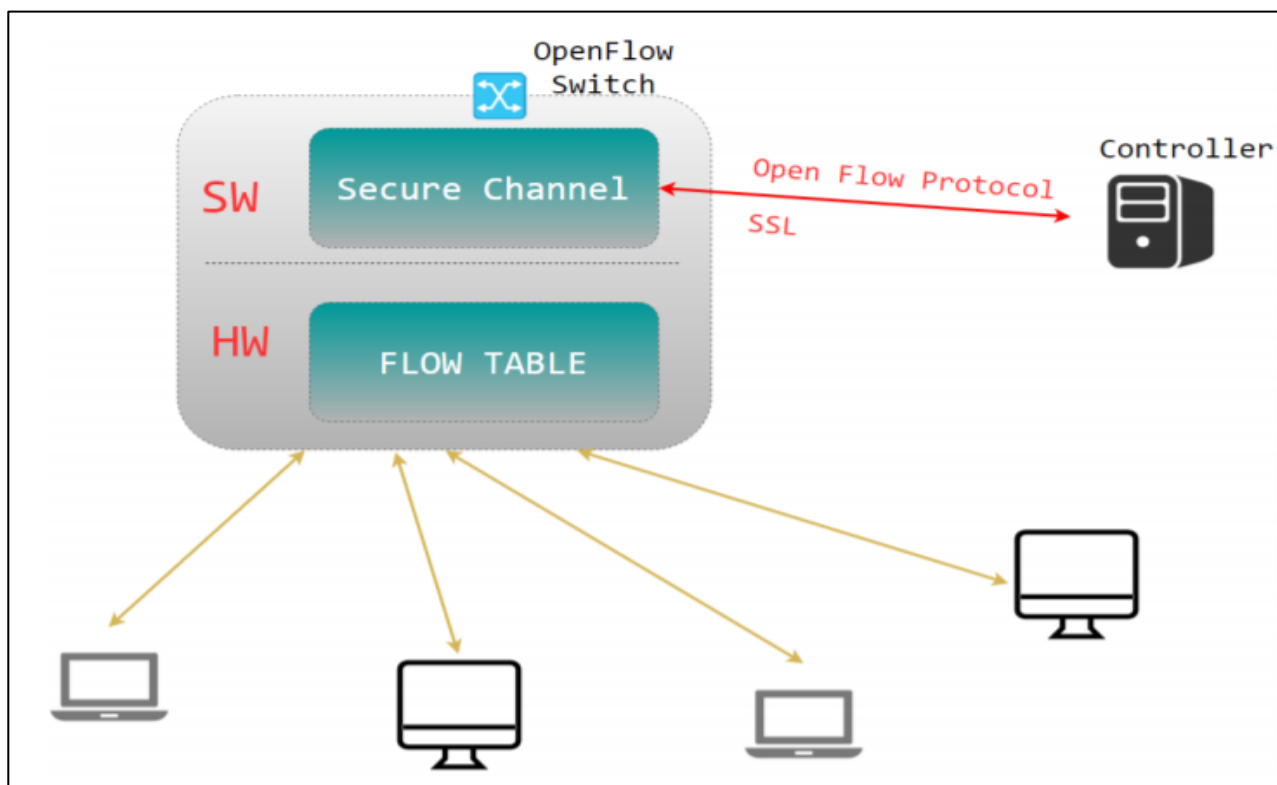
Ako všetky iné protokoly a komunikácie je aj OpenFlow náchylná na možné útoky. Medzi najtypickejšie útoky pre tento typ siete patria[12] [13]:

- man-in-the-middle útok
- single point útok a zlyhanie
- problémy spojené s programovacími a komunikačnými kanálmi

2.3. SDN forwardery

Preposielacie zariadenie pozostáva z dvoch častí [14]:

- prietokovej tabuľky (flow table) obsahujúcej záznam a akciu na prijímanie aktívnych tokov
- abstrakčnej vrstvy, ktorá bezpečne komunikuje s riadiacou jednotkou (kontrolérom) o nových vstupoch, ktoré sa v danom momente nenachádzajú v tokovej tabuľke.



Obrázok 3. Zobrazenie SDN prepínača a jeho vnútornej štruktúry pri komunikácii s kontrolérom [14]

2.3.1. Prepínač

Prepínače v SDN často predstavujú preposielací hardvér, ktorý je prístupný cez otvorené rozhranie. V sieti OpenFlow sa prepínače delia do dvoch základných skupín [14]:

- čisté (pure) – nemajú žiadnu funkcionality a spoliehajú sa len na kontrolér, ktorý rozhoduje o preposielaní paketov
- hybridné (hybrid) – okrem „tradičných“ protokolov a operácii podporujú aj OpenFlow.

OpenFlow prepínač pozostáva z jednej alebo viacerých tabuliek tokov, ktoré ukladajú záznamy pre vyhľadanie alebo presmerovanie paketov. [14]

Po príchode paketov na OpenFlow prepínač sa hlavička paketu extrahuje a porovná s políčkom zhody (match field) z tabuľky tokov. Ak sa zhoduje hlavička paketu s políčkom v tabuľke, tak prepínač použije príslušnú sadu inštrukcií spojenú s daným tokom. V prípade, že sa nenájde žiadna zhoda s tabuľkou tokov, tak nasledujúca akcia prepínača bude závisieť od inštrukcií definovaných v tabuľke chýbajúcich tokov (table-miss flow entry). [14]

Akcie obsiahnuté v tabuľke chýbajúcich tokov sú napr. zahodenie paketu, odoslanie paketu kontroléru cez OpenFlow kanál. [14]

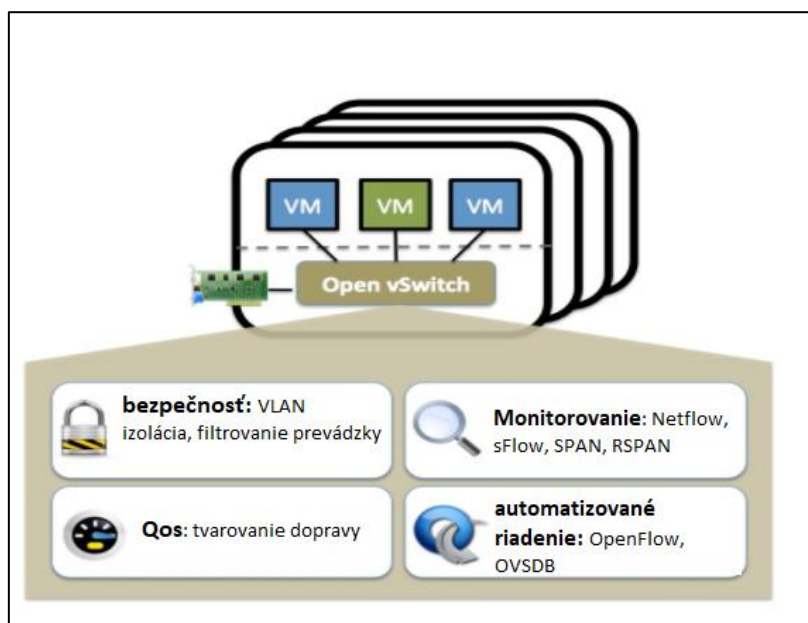
Tabuľka 1. Vlastnosti OpenFlow prepínačov [14]

Soft. prepínač	Implementácia	Opis	Ryu	Verzia
Open vSwitch	C/Python	Podporuje štandardný manažment rozhraní a umožňuje programové rozšírenie a riadenie funkcií preposielania. Je možné ho aplikovať do štandardných prepínačov. [1]	✓	v1.0
Pantou/OpenWRT	C	Mení komerčný bezdrôtový smerovač alebo prístupový bod na OpenFlow prepínač. [1]	X	v1.0
OFSwitch13	C/C++	Kompatibilný s OpenFlow 1.3. [1]	X	v1.3
Linc	Erlang	LINC je navrhnutý tak, aby pracoval v rôznych operačných systémoch	✓	v1.2
Indigo	C	OpenFlow implementácia na fyzických prepínačoch využívajúca ich funkcie na spustenie OpenFlow. [1]	✓	v1.0

2.3.2. Open vSwitch (OVS)

Open vSwitch je kvalitný viacvrstvový virtuálny prepínač. Je navrhnutý tak, aby umožňoval masívnu automatizáciu siete pomocou programového rozšírenia a zároveň podporoval štandardný manažment rozhraní a protokolov. Open vSwitch podporuje OpenFlow verzie 1.0, 1.1, 1.2, 1.3. [15]

Open vSwitch má podporu pre štandardizované fyzické servery, linuxové distribúcie a Microsoft Windows. [16]



Obrázok 4. Vlastnosti Open vSwitch [17]

2.3.3. Pantou/OpenWRT

Pantou je založená na verzii BackFire 10.03 OpenWRT (Linux 2.6.32). Tento upravený balík OpenWRT obsahuje virtuálny prepínač, ktorý funguje ako aplikácia umožňujúca transformáciu bezdrôtového smerovača na prepínač. [15]

2.3.4. OFSoftSwitch13

Modul OFSoftSwitch13 poskytuje podporu pre protokol OpenFlow verzie 1.3, čím prináša prepínač a rozhranie kontroléra na simulátor ns-3. Komunikácia medzi kontrolérom a prepínačom je realizovaná cez protokol ns-3. Modul využíva externý OpenFlow 1.3 softvérový prepínač, ktorý je kompilovaný ako knižnica. [17]

2.3.5. Indigo Virtual Switch (IVS)

Projekt Indigo podporuje OpenFlow protokol na fyzických prepínačoch. Je navrhnutý pre vysoký výkon s nízkou administráciou. Indigo projekt podporuje len OpenFlow verziu 1.0. [15]

2.3.6. Linc

Linc je open source projekt, ktorý podporuje protokoly OpenFlow verzie 1.2 a 1.3. LINC je navrhnutý tak, aby využíval všeobecne dostupné komody, x86 hardvér a pracoval v rôznych operačných systémoch (Linux, Mac, Windows,...). [15] Projekt je momentálne pozastavený.

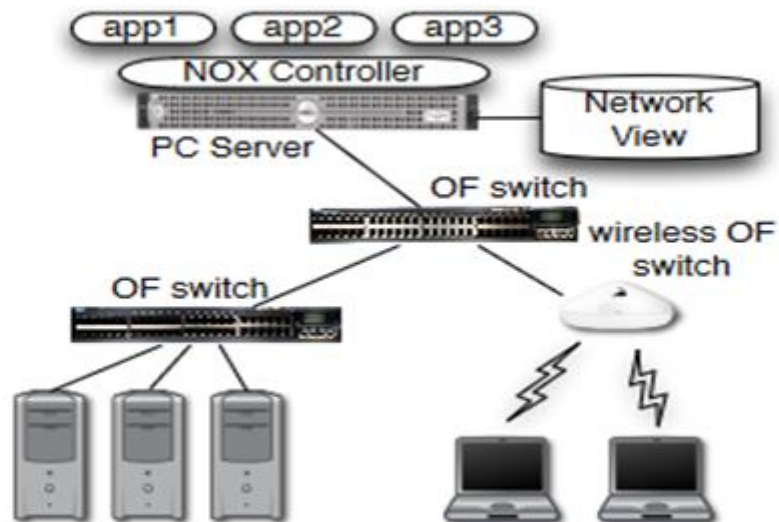
2.4. SDN kontroléry

OpenFlow kontrolér je typ SDN kontroléra, ktorý používa OpenFlow protokol. OpenFlow kontrolér používa OpenFlow protokol aby spojil a konfiguroval sieťové zariadenia, ako smerovače a prepínače, na nájdenie najlepšej cesty v premávke. SDN kontroléry teda uľahčujú riadenie siete a dokážu riadiť celú komunikáciu medzi aplikáciami a zariadeniami, tak aby sa čo najefektívnejšie upravil tok premávky, tak ako je v danom momente potrebné. Keď nie je riadiaca rovina implementovaná vo firmvéri, ale je implementovaná v softvéri, administrátori dokážu manažovať sieť oveľa jednoduchšie, viac dynamicky a s lepšou granularitou. OpenFlow kontrolér teda tvorí centrálnu kontrolnú jednotku v sieti, ktorá riadi všetku premávku, všetky zariadenia v sieti vykonávajú akcie tak ako od nich požaduje kontrolér a podporuje OpenFlow. OpenFlow je najpopulárnejší štandardný protokol používaný v SDN. [1] [2] [3]

2.4.1. NOX

NOX bol prvým open source kontrolérom pre OpenFlow, vydaným v roku 2008. NOX je open source OpenFlow kontrolér, ktorý poskytuje kontrolu nad sieťou s OpenFlow prepínačmi. Podporuje aplikácie v jazyku Python a C++, samotný kontrolér je napísaný v jazyku C++, reprezentácia pohľadu na sieť je formou indexovaných hashovacích tabuliek s vyrovnávacou pamäťou pre zrýchlenie toku v sieti. Jeden kontrolér vie spracovať 100 000 tokových požiadaviek za sekundu. NOX má len jednu globálnu dátovú štruktúru, pohľad na sieť, ktorý sa mení dosť pomaly na to aby bol udržiavaný centrálny a to aj vo veľmi veľkých sieťach. Informácie z tejto dátovej štruktúry slúžia na rozhodovanie sa a využívajú rozhodovanie v riadení systému. [4][5]

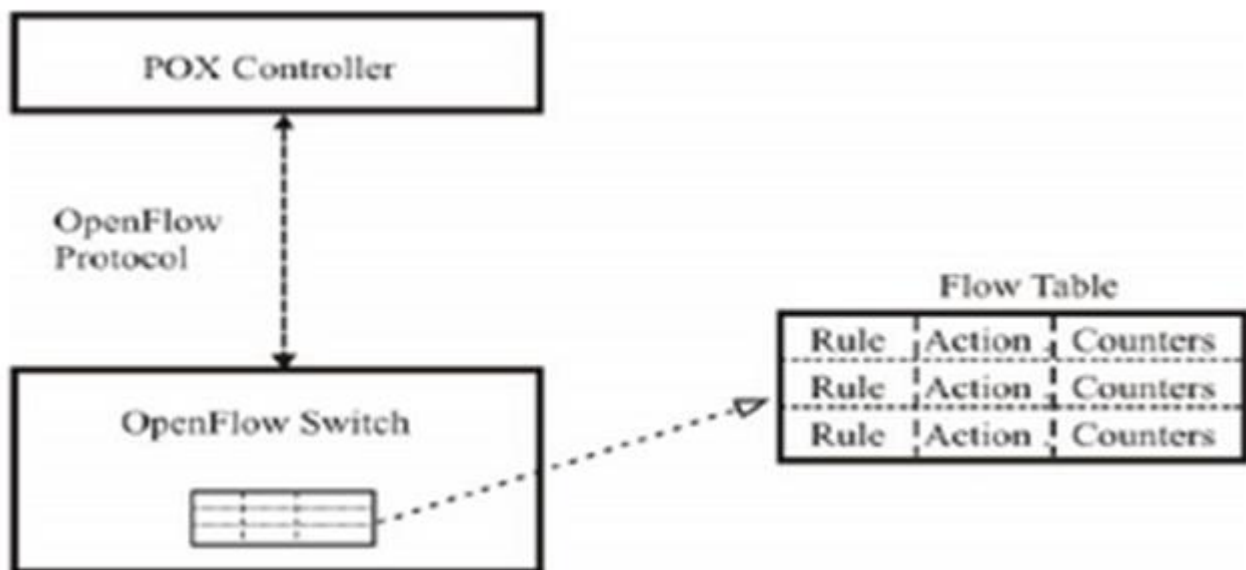
NOX používa rovnakú reprezentáciu. Pre každý paket, ktorý sa zhoduje v poli hlavička, je aktualizované pole počítačiel a sú vykonané príslušné akcie. Ak sa paket zhoduje s viacerými tokovými vstupmi, vyberie sa vstup s najväčšou prioritou. NOX podporuje OpenFlow verzie 1.3. [4] [5]



Obrázok 5. NOX kontrolér[4]

2.4.2. POX

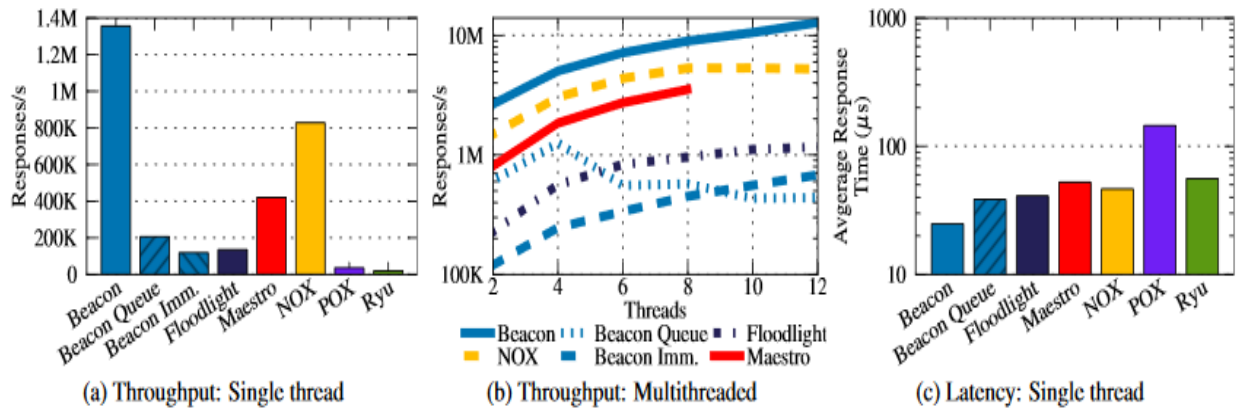
POX je open source kontrolér pre vývoj SDN aplikácií. POX kontrolér umožňuje efektívnu implementáciu OpenFlow protokolu do SDN. Tcp dump packet capture je nástroj, ktorý dokáže zachytiť pakety tečúce v sieti medzi kontrolérom a OpenFlow zariadeniami. POX kontrolér je založený na predošlom NOX kontroléri. Je napísaný v jazyku Python a je menej komplexný ako NOX. POX sa používa najmä na rýchle prototypovanie jednoduchších aplikácií pre SDN. Môže byť skombinovaný s reálnym hardvérom a je dobre kompatibilný s Mininetom, nemá však GUI rozhranie. Podporuje len však OpenFlow verzie 1.0. [6]



Obrázok 6. POX kontrolér[6]

2.4.3. Beacon

Beacon je OpenFlow kontrolér založený na Java a frameworku Spring. Bol vyvinutý najmä pre pohodlnejšie programovanie programátorov, rýchlejšiu kompiláciu a lepší výkon. Pri jazyku C++/Python kompilácia aj jednoduchších sietí mohla trvať aj 10minút, čo pri vývoji robilo problémy. Beacon dokáže pracovať tak aby v reálnom čase mohla začať alebo ukončiť aktuálnu alebo novú aplikáciu. Beacon na komunikáciu používa Java knižnicu OpenFlowJ. Výhodou Beaconu je aj webové užívateľské rozhranie a stabilita. Podporuje však taktiež len OpenFlow verzie 1.0. [5]



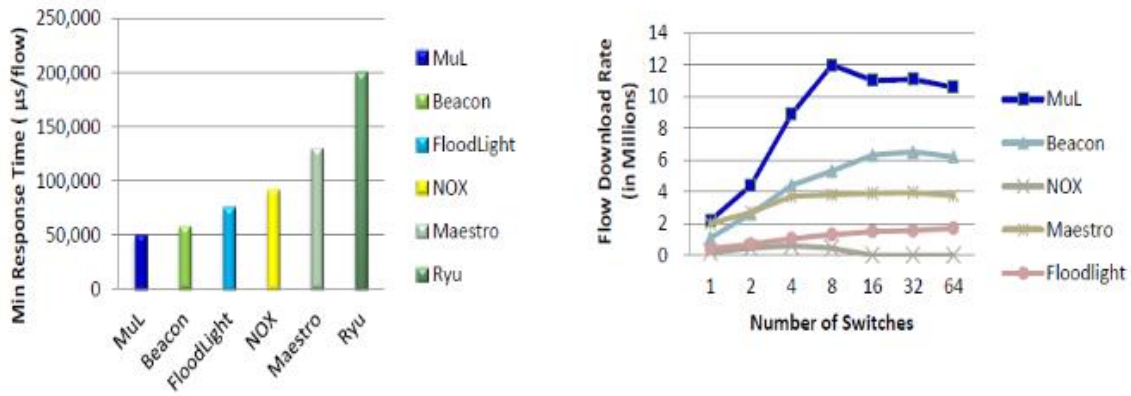
Obrázok 7. Porovnanie kontrolérov [5]

2.4.4. Floodlight

Floodlight je OpenFlow kontrolér založený na Java a frameworku Netty. Floodlight obsahuje modul OFSwitchManager pomocou, ktorého komunikuje s pripojenými switchami. Od ostatných kontrolérov sa líši najmä tým, že je synchronný. Návrh Floodlightu je však príliš náročný na dnešné produkčné prostredie, preto nevieme využiť jeho plný potenciál. Výhodou Floodlightu je naozaj jednoduchá inštalácia s minimálnymi závislosťami. Podporuje OpenFlow verzie 1.0, avšak Floodlight-plus podporuje aj verziu 1.3. [7]

2.4.5. MUL

MUL je OpenFlow kontrolér založený na jazyku C, vďaka ktorému dosahuje najlepšie výkonnostné hodnoty. Okrem jazyka C MUL kontrolér ponúka dve funkcie FirmFlow a FlexPlug, ktoré zvyšujú najmä bezpečnosť siete a flexibilitu siete. MUL podporuje najnovšiu verziu OpenFlow. [8]



Obrázok 8. Porovnanie kontrolérov[8]

2.4.6. Maestro

Maestro je OpenFlow kontrolér založený na Java a je prvým kontrolérom, ktorý umožňuje paralelizmus, aby zabezpečil čo najlepšiu výkonnosť. Okrem dobrého výkonu a paralelizmu neposkytuje žiadne veľké výhody. Podporuje OpenFlow verzie 1.0. [9]

2.4.7. Iné

Ďalšími alternatívami kotrolérov môžu byť aj Trema, Flowe a Nette, no tie žiaľ v dnešnej dobe ešte nespĺňajú podmienky plnohodnotných kontrolérov, pretože sú ešte vo vývoji a je na nich obmedzená funkcionlita. [2]

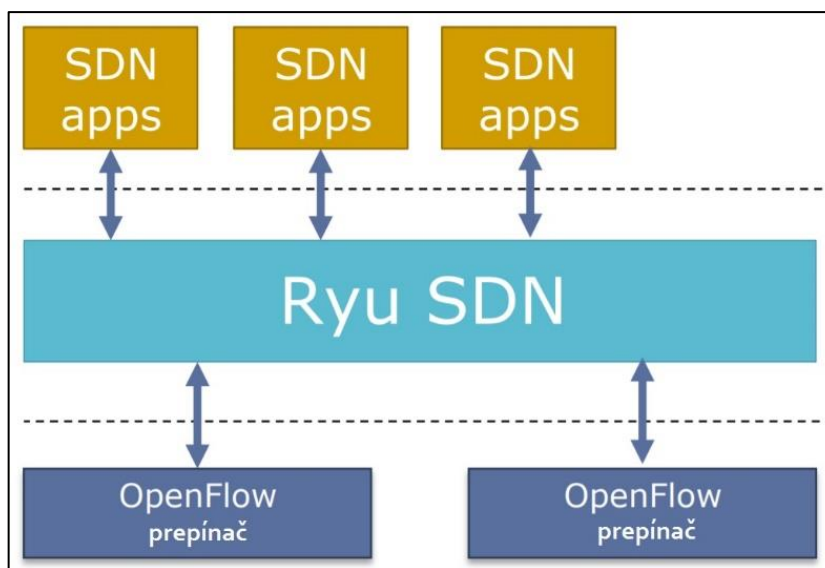
Use-Cases	Trema	Nox/Pox	RYU	Floodlight	ODL	ONOS***
Network Virtualizaiton by Virtual Overlays	YES	YES	YES	PARTIAL	YES	NO
Hop-by-hop Network Virtualization	NO	NO	NO	YES	YES	YES
OpenStack Neutron Support	NO	NO	YES	YES	YES	NO
Legacy Network Interoperability	NO	NO	NO	NO	YES	PARTIAL
Service Insertion and Chaining	NO	NO	PARTIAL	NO	YES	PARTIAL
Network Monitoring	PARTIAL	PARTIAL	YES	YES	YES	YES
Policy Enforcement	NO	NO	NO	PARTIAL	YES	PARTIAL
Load Balancing	NO	NO	NO	NO	YES	NO
Traffic Engineering	PARTIAL	PARTIAL	PARTIAL	PARTIAL	YES	PARTIAL
Dynamic Network Taps	NO	NO	YES	YES	YES	NO
Multi-Layer Network Optimization	NO	NO	NO	NO	PARTIAL	PARTIAL
Transport Networks - NV, Traffic-Rerouting, Interconnecting DCs, etc.	NO	NO	PARTIAL	NO	PARTIAL	PARTIAL
Campus Networks	PARTIAL	PARTIAL	PARTIAL	PARTIAL	PARTIAL	NO
Routing	YES	NO	YES	YES	YES	YES

Obrázok 9. Porovnanie kontrolérov[10]

2.5. Kontrolér Ryu

Ryu kontrolér je jeden z najznámejších a zároveň najpoužívanějších kontrolérov v SDN sieťach. Jeho názov vychádza z japončiny, kde v dvoch významoch znamená tok, resp. Japonský drak. Ryu je voľne šíriteľný sieťový operačný systém – programovacie sieťové rozhranie (logicky centralizované), voľne dostupné pod licenciou Apache 2. Najnovšia verzia k písaniu tohto dokumentu je 4.18. [24] [25] [26]

Ryu poskytuje veľmi silný pomer medzi jeho výhodami a nevýhodami, v prospech výhod. Ryu vďaka svojmu rozšíreniu a štandardu kontroléra má širokú základňu aktívnych používateľov, ktorí vytvárajú masívny zdroj informácií a spätných väzieb pre vývojárov, preto je stále v aktívnom vývoji a udržiavaní kódu. Taktiež je považovaný za formálny štandard pre OpenStack, čo je voľne šíriteľný softvér pre stavanie privátnych a verejných cloudových riešení. Medziiným poskytuje konzistentnú topologizáciu na druhej vrstve OSI modelu nezávisle od tej fyzickej. [24] [25] [26]



Obrázok 10. Štruktúra SDN z hľadiska vrstiev[24] [25] [26]

Ryu si stavia medzi jej hlavne piliere výhod tri charakteristiky: kvalita kódu, funkcionálna a použiteľnosť. Podporuje niekoľko protokolov pre správu sieťových zariadení, tými sú napr. Netconf, OF-config, no primárne OpenFlow vo verziách až po aktuálnu 1.5 (k písaniu dokumentu) spolu s rozšíreniami Nicira. Napriek množstvu typových produktov na trhu je principiálne nezávislý na výrobcovi a tak schopný spolupráce. Ryu je najmä kvôli prvému bodu vhodná pre rozsiahle produkčné prostredia. Na druhej strane, práve implementáciou kódu, ktorá je v plnej miere

v programovacom jazyku Python, je oproti ostatným riešeniam značne pomalšia. Z hľadiska kompatibility, pracuje s verziami jazyka Python 2.7, 3 a 3.4. [24] [25] [26]

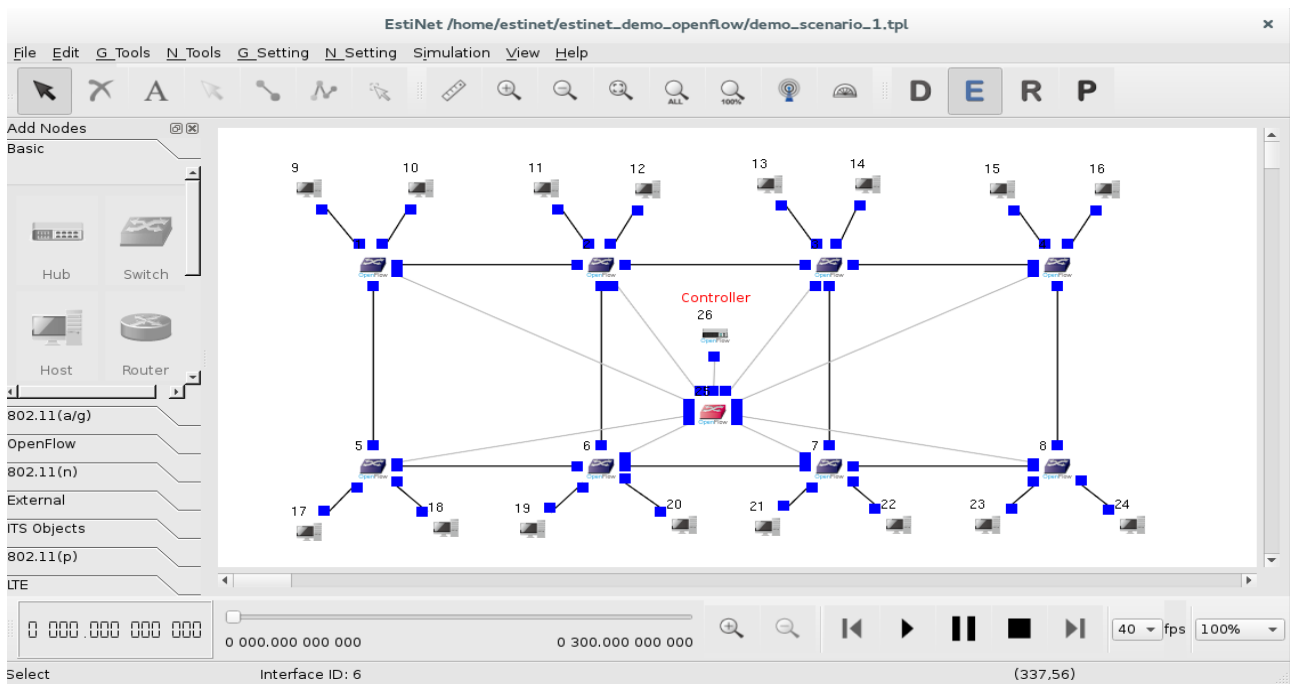
Ryu ako softvér pozostáva z určitých častí – nazývané komponenty. Komponent je samostatný blok, ktorý slúži ako rozhranie pre kontrolu a vytváranie udalostí. Takýchto komponentov poskytuje Ryu v základe niekoľko. Tie existujúce sa dajú modifikovať, taktiež je možné pridávať a kombinovať nové a komunikácia prebieha na základne nezávislých správ. Vstavané komponenty sú v jazyku Python a pozostávajú z vlákien a procesov operačného systému. Dodatočné časti môžu byť naviazané aj na už existujúce, musí byť však dodržaná metodika správ. Medzi dostupné komponenty patria napríklad OF-wire, Topology, VRRP, OF REST, Endpoint alebo Stats. Zoskupením niektorých častých častí pre potreby efektivity sú vytvorené knižnice. Tie obsahujú metódy, ktoré sú volané priamo komponentmi. Takéto knižnice sú Netconf, OF-conf, sFlow, NetFlow, či OVSDb JSON. [24] [25] [26]

2.6. Simulačné/emulačné prostredia

Existuje niekoľko vývojových platforiem, pomocou ktorých je možné simulovať SDN projekty. Umožňuje vytváranie a vykonávanie experimentov. [15]

2.6.1. EstiNet

EstiNet umožňuje simuláciu niekoľkých sieťových protokolov. Jeden z nich je OpenFlow protokol. EstiNet má kvalitné grafické vlastnosti pri vizualizácii experimentu. Vizualizuje animáciu paketov plus grafická reprezentácia jednotlivých uzlov v sieti. [15]



Obrázok 11. Prostredie EstiNet[15]

2.6.2. Ns-3

Ns nemá kompatibilitu s novou verzou OpenFlow protokolu. Rovnako nepodporuje openflow kontroléry: NOX, POX alebo Floodlight bez modifikácií. [15]

2.6.3. Trema

Trema poskytuje všetky prostriedky pre vývoj a experimentovanie SND siete. Má integrované testovacie a debugovacie prostredie, ktoré monitoruje, diagnostikuje celú sieť (Trema shark, Wireshark plug-in). Avšak neposkytuje grafické rozhranie a samotný simulátor je napísaný v jazyku C a Ruby, čo limituje popularitu danej platformy. [15]

2.6.4. Mininet

Mininet je sieťový emulátor. Umožňuje vytvárať koncové zariadenia, prepínače, smerovače, a linky medzi nimi na jednom Linuxovom kernely. Mininet host sa správa rovnako ako reálna mašina a je možné sa naň pripojiť pomocou SSH. Čo sa týka OpenFlow kontrolérov, Mininet je veľmi flexibilný a umožňuje pridať do simulácie množstvo typov kontrolérov. [15]

Nevýhody Mininetu[15]:

- Je kompatibilný jedine s operačným systémom Linux
- Neposkytuje siete s rýchlosťou 100Gbps.

3. Generátory premávky

V mininete beží takmer každý linuxový program. Takže je možné využiť takmer akýkoľvek klientský alebo serverový program pre tvorbu premávky (ping, iperf, wget, curl, netperf, netcat,...). Premávku je možné zachytiť pomocou programov wireshark a tcpdump. Vygenerovať pakety je tiež možné pomocou jazyka python za využitia scapy. [18]

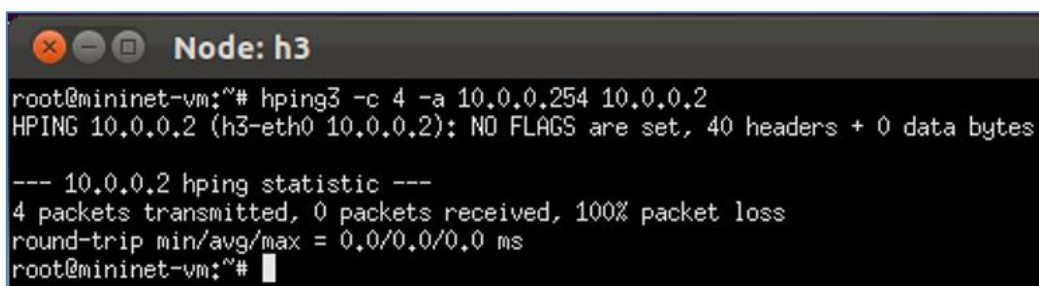
Kapitola opisuje a porovnáva voľne dostupné generátory premávky, akými sú napr. : Hping, Ostinato, Scapy, packETH.

3.1. Hping

Hping je jedným z najznámejších a bezplatných nástrojov pre tvorbu premávky. Umožňuje zostaviť vlastné ICMP, UDP, TCP a Raw IP pakety. Nástroj bol v minulosti používaný hlavne na bezpečnosť, ale je možné ho využívať aj v iných smeroch ako: [19]

- Testovanie firewall
- Rozšírené skenovanie portov
- Testovanie siete pomocou rôznych protokolov

Nástroj nemá používateľské rozhranie, teda je možné ho spúšťať len cez príkazový riadok. Hping má traceroute režim a tiež schopnosť odosielania súborov medzi krytým kanálom. [19]



```
root@mininet-vm:~# hping3 -c 4 -a 10.0.0.254 10.0.0.2
HPING 10.0.0.2 (h3-eth0 10.0.0.2): NO FLAGS are set, 40 headers + 0 data bytes

--- 10.0.0.2 hping statistic ---
4 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@mininet-vm:~#
```

Obrázok 12. Príklad výsledku použitia hping3 príkazu

Hping nie je v štandardnej inštalácii a treba ho doinštalovať pomocou príkazu: `sudo apt-get install hping3`

Základné testovanie

Hping3 umožňuje testovať všetky vyššie spomenuté prípady, umožňuje posielanie súborov, aj keď sa nepodarilo nájsť posielanie videí. Hping3 umožňuje testovanie viacerých funkcionalít naraz ale z rôznych koncových zariadení či prepínačov. Tiež je možné otestovanie rôznych prípadov napadnutí.

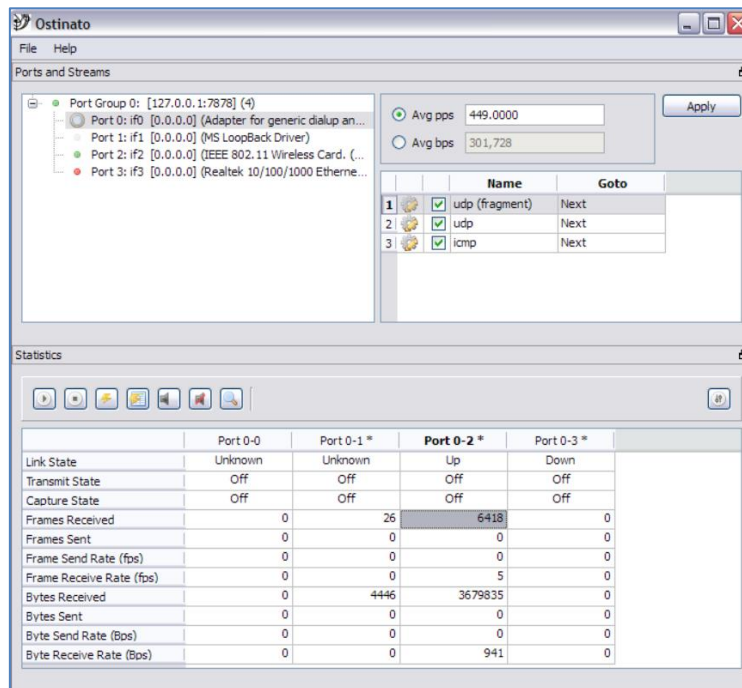
3.2. Ostinato

Ostinato je voľne dostupný paketový generátor a analytický nástroj. Na rozdiel do Hping využíva používateľské rozhranie, ktoré umožňuje jednoduché používanie. [18]

Nástroj podporuje najbežnejšie protokoly:

- Ethernet/802.3/LLC SNAP
- VLAN (aj QinQ)
- ARP, IPv4, IPv6, IP Tunneling
- TCP, UDP, ICMP
- každý textový protokol (http, SIP, RSTP,...)

Pomocou programu je možné ľahko upraviť ľubovoľné pole každého protokolu. [18]

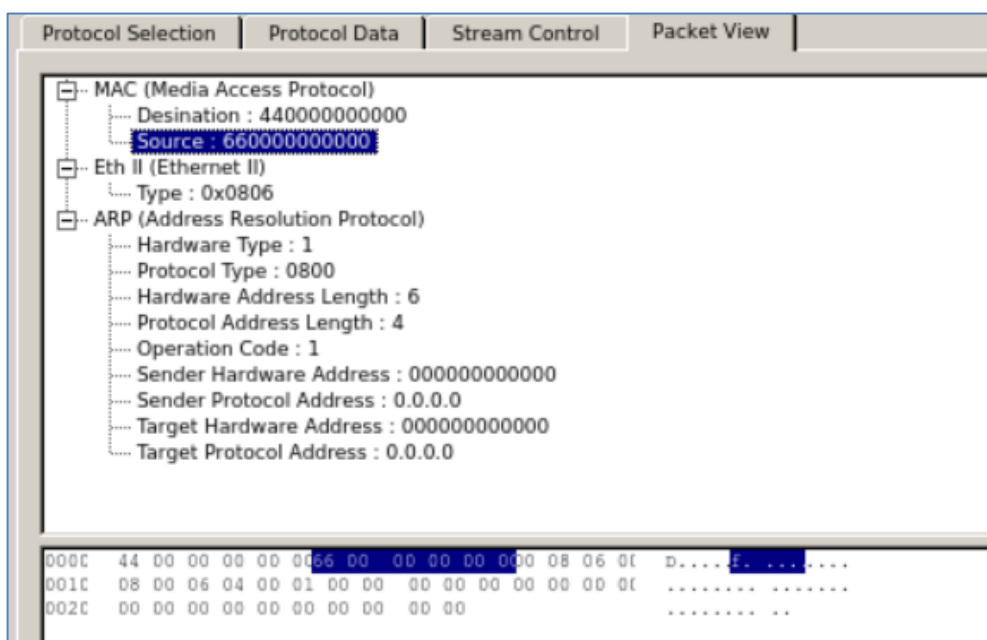


Obrázok 13. Používateľské rozhranie

Ostinato podobne ako Hping treba stiahnuť pomocou príkazu: `sudo apt-get install ostinato`. Princíp spúšťania nástroja na virtuálnom zdroji je popísaný v [20].

Základné testovanie

Pri ostinato je pekné, že je možné vidieť štatistiky jednotlivých portov a tiež je možné v ňom si rozbaľiť pakety. Tiež má veľmi dobrú používateľskú príručku¹. Od verzie 6.0 ostinato podporuje Python skriptovanie, vďaka ktorému je možné rozšíriť ostinato o ďalšie protokoly. Ostinato umožňuje otváranie a zmenu PCAP súborov a tiež je možné ho spúšťať a testovať na viacerých hostoch naraz. Nikde sa ale nenašla možnosť preposielania videa.



Obrázok 14. Rozbalenie paketu pomocou ostinato

3.3. Scapy

Scapy je ďalší nástroj na vytváranie paketov, ktorý bol napísaný v Pythone. Môže dekódovať alebo vytvárať pakety pre širokú škálu protokolov. Môže vykonávať rôzne úlohy vrátane skenovania, vyhľadávania, skúšania alebo testovania siete. [18]

Na rozdiel od iných nástrojov umožňuje aj odosielanie neplatných rámcov, vytváranie si vlastných 802.11 rámcov, kombinovanie rôznych technológií (VLAN hopping + ARP cache poisoning, VOIP na šifrovanom kanále WEP,...). [21]

¹ <https://userguide.ostinato.org/Quickstart.html>

Princíp fungovania Scapy je popísaný v [22].

Základné testovanie

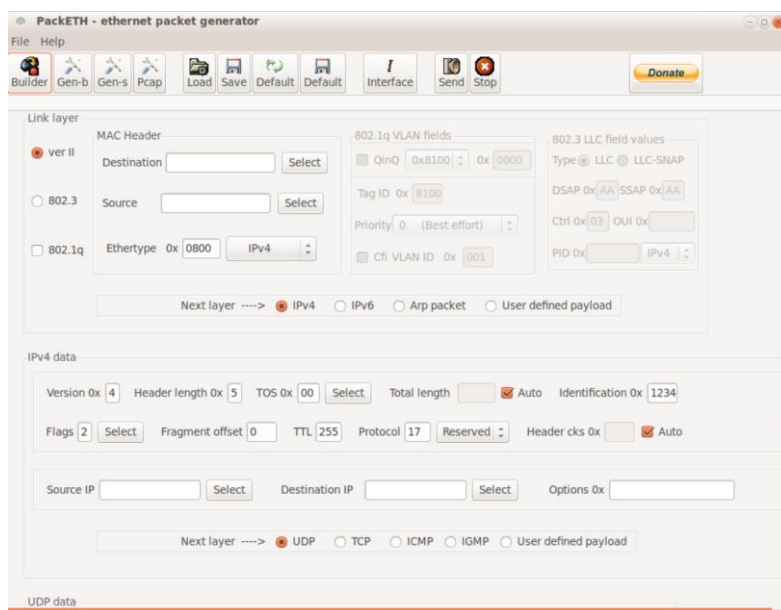
Príkazom `sudo scapy` spustíme scapy rozhranie. Nevýhodou je, že je potreba manuálne si tvoriť pakety na posielanie. Výhodou zas je, že je možné si testovacie scenáre ukladať do Python skriptov a tie následne testovať. Do skriptu si je možné zvoliť aj typ prenášania (video, obrázok).

```
root@mininet-vm:~/mininet/custom# sudo scapy
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
INFO: No IPv6 support in kernel
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.2.0)
>>> █
```

Obrázok 15. Scapy rozhranie

3.4. packETH

PackETH je linuxový nástroj pre vytváranie paketov pomocou používateľského rozhrania. Umožňuje rýchlo vytvoriť a odoslať sekvenciu paketov. Rovnako ako ostatné nástroje podporuje rôzne protokoly. Je možné tiež nastaviť počet paketov a oneskorenie medzi nimi. [18]



Obrázok 16. Používateľské rozhranie

Nástroj podporuje vytvorenie a odoslanie akéhokoľvek ethernetového paketu. Podporované protokoly sú [23]:

- ethernet II, ethernet 802.3, 802.1q, QinQ, užívateľom definovaný ethernetový rámec
- ARP, IPv4, IPv6
- UDP, TCP, ICMP
- RTP
- JUMBO rámce

Základné testovanie

Generátor je možné stiahnuť pomocou príkazu `sudo apt-get install packeth`. A spustiť ho je možné pomocou príkazu `packeth`, kedy sa zobrazí používateľské rozhranie. Cez rozhranie je možné posielat' pakety buď po jednom alebo aj viacej naraz. Tiež je možné si v súbore uložiť IP a z daného súboru ich načítavať (žiaľ súbor musí mať pevne definovaný formát). S tým, že je možné posielat' aj minimálne pakety obsahujúce zvuk².

Pri tomto generátore ale nastal problém, že aj keď je možnosť posielanie len jedného paketu, pošle sa ich omnoho viac naraz, ako je znázornené v obrázku nižšie.

8698	60.71099500€	66:00:00:00:00:00	44:00:00:00:00:00	ARP	60	Who has 192.168.0.4?	Tell 192.168.0.6
8699	60.71106900€	66:00:00:00:00:00	44:00:00:00:00:00	ARP	60	Who has 192.168.0.4?	Tell 192.168.0.6
8700	60.71107100€	66:00:00:00:00:00	44:00:00:00:00:00	ARP	60	Who has 192.168.0.4?	Tell 192.168.0.6
8701	60.71110400€	66:00:00:00:00:00	44:00:00:00:00:00	ARP	60	Who has 192.168.0.4?	Tell 192.168.0.6
8702	60.71110500€	66:00:00:00:00:00	44:00:00:00:00:00	ARP	60	Who has 192.168.0.4?	Tell 192.168.0.6
8703	60.71112700€	66:00:00:00:00:00	44:00:00:00:00:00	ARP	60	Who has 192.168.0.4?	Tell 192.168.0.6
8704	60.71112800€	66:00:00:00:00:00	44:00:00:00:00:00	ARP	60	Who has 192.168.0.4?	Tell 192.168.0.6
8705	60.71120300€	66:00:00:00:00:00	44:00:00:00:00:00	ARP	60	Who has 192.168.0.4?	Tell 192.168.0.6
8706	60.71126900€	66:00:00:00:00:00	44:00:00:00:00:00	ARP	60	Who has 192.168.0.4?	Tell 192.168.0.6
8707	60.71131400€	44:00:00:00:00:00	66:00:00:00:00:00	ARP	42	192.168.0.4 is at 44:00:00:00:00:00	

Obrázok 17. Zobrazenie ARP pre PackETH generátor

² <https://packeth.wordpress.com/2014/09/04/sending-rtp-stream-with-packeth/>

4. QoS z hľadiska poskytovania služieb na sieti

4.1. VOIP

Voice over IP, jedná sa o hovorovú komunikáciu cez internet. [27]

4.1.1. Latencia

Ľudia, ktorí bežne telefonujú zachytia oneskorenie 250ms alebo viac. ITU-T G.114 odporúčajú maximálnu latenciu 150ms jedným smerom. Takže sieť by mala mať porovnateľne menej ako 150ms latenciu. Rôzni poskytovatelia a ich maximálny limit pre latenciu [28]:

- Axiowave SLA 64ms max
- Internap SLA 45ms max
- Qwest SLA 50ms max
- Verio SLA 55ms
- Max 150 ms

4.1.2. Odchýlka

Rôzni poskytovatelia a ich maximálny limit pre odchýlku [29]:

- Axiowave SLA 0.5ms max
- Internap SLA 0.5ms max
- Qwest SLA 2ms max
- Verio SLA 0.5ms priemerne, nesmie prekročiť 10ms viac ako 0.1 percento času
- Viterla SLA 1ms max
- max 30ms

4.1.3. Stratovosť paketov

VOIP nie je tolerantné ku strate paketov. Už jedno percentné straty výrazne znižujú VOIP volanie používajúce G.711, a iné kompresné algoritmy tolerujú ešte menšie straty. Ideálne sú žiadne straty paketov v VOIP.

Rôzni poskytovatelia a ich maximálny limit pre stratovosť paketov [29]:

- Axiowave SLA 0 max
- Internap SLA 0.3 max
- Qwest SLA 0.5 max
- Verio SLA 0.1 max

4.2. Video interaktivita

Interaktívna komunikácia pomocou videa. Napríklad cez službu Skype, keď máme video konferenciu.

4.2.1. Latencia

- Nie viac ako 150ms

V živej video konferencii alebo pri hraní hry, sa považuje pod 100ms ako nízka latencia, pretože ľudské oko ju nevie ešte výrazne zaznamenať. Pri komunikácii počítača s obsahom videa, je normálne uvažovať ešte menšie ako 30, 10 alebo dokonca pod 1 ms. [27]

4.2.2. Stratovosť paketov

Strata nie viac ako 1 percento. [27]

4.2.3. Odchýlka

Nie viac ako 30ms. [27]

4.2.4. Šírka pásma

Je 20 a viac percent z potrebnej prenosovej rýchlosti. Napríklad 384-kbps video konferenčné stretnutie potrebuje 460-kbps garantovanej šírky pásma. [28]

4.3. Video stream

Jednoduchý stream videa z Internetu, napríklad z portálu Youtube.

4.3.1. Latencia

Napríklad 100ms/33.3ms na 1 „frame“ = strata 3 „frame“. Nemala by byť väčšia ako 4 až 5 sekúnd. [28]

4.3.2. Stratovosť paketov

Strata by nemala byť väčšia ako 5 percent. [28]

4.3.3. Odchýlka

Nemá žiadne signifikantné obmedzenia. [28]

4.3.4. Šírka pásma

Záleží od rozlíšenia a kódovania. [28]

QCI	Bearer Type	Application Example	Packet Delay	Packet Loss	Priority
1	GBR	Conversational VoIP	100ms	10^{-2}	2
2		Conversational Video (Live Streaming)	150ms	10^{-3}	4
3		Non-Conversational Video (Buffered Streaming)	300ms	10^{-6}	5
4		Real Time Gaming	50ms	10^{-3}	3
5	NON-GBR	IMS Signaling	100ms	10^{-6}	1
6		Voice, Video, Interactive Games	100ms	10^{-3}	7
7		Video (Buffered Streaming)	300ms	10^{-6}	6
8		TCP apps (web, email, ftp)			8
9		Platinum vs. gold user			9

Obrázok 18. Stratovosť a oneskorenia paketov VoIP, Video, Hry, TCP aplikácie [27] [28] [29]

4.4. Dáta

Nad dátami pre QoS sú štandardy dosť všeobecné. Majú mať dostatočne veľkú šírku pásma, ale nemajú veľmi obmedzovať celkovú priepustnosť siete. V zozname sa nachádzajú na konci, pretože nemajú veľmi špecifické nároky na QoS.

5. Testovanie

Vytvorenie testovacej topológie prebiehalo cez inicializačný skript v jazyku python, priebeh testovania, spolu s konkrétnymi krokmi sa nachádza v časti technická dokumentácia C – Priebeh testovania.

5.1. Testovací scenár č.1 (Hlavný testovací scenár)

Cieľom testovacieho scenára, je zistenie cesty, ktorú si kontrolér zvolil ako hlavnú pri preposielaní paketov a následne overenie, či cesta sa zmení v prípade vypnutia linky alebo zmenenia šírky pásma.

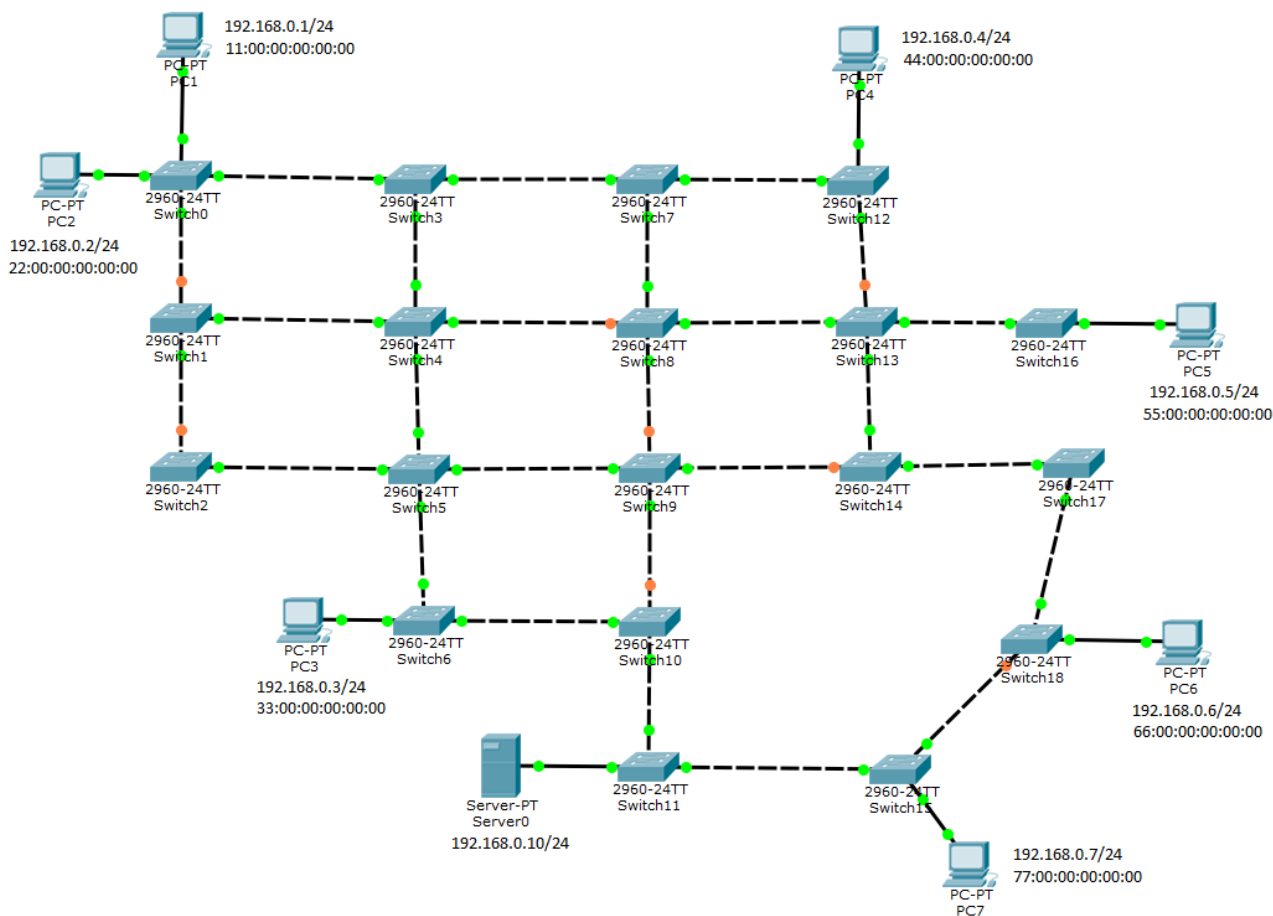
5.1.1. Testovacie prostredie

Operačný systém: Linux

OpenFlow prepínač: Open vSwitch

OpenFlow kontrolér: Ryu kontrolér

5.1.2. Testovacia topológia



Obrázok 19. Testovacia topológia č.1

5.1.3. Zhodnotenie

1. Sieť je zhotovená podľa topológie
2. Ping sa úspešne podaril na všetky zariadenia, aj keď kontroleru dlhšie trvá kým sa prepojí. V niektorých prípadoch dokonca neprešiel ping na zopár koncových zariadení a bolo treba reštartovať kontrolera a aj topológiu.
3. Pomocou príkazov: `sudo ovs-ofctl dump-flows sxx` a `sudo ovs-ofctl show sxx` (xx predstavujú poradové číslo prepínača) a ryu používateľského rozhrania ³ som zostavila cestu z h1 do h5. Cesta vedie cez nasledovné prepínače:

s00 – s03 – s07 – s08 – s13 – s16

³ `cd ryu`
`ryu-manager --observe-links ryu/app/gui_topology/gui_topology.py`

4. Pomocou príkazu `sudo ovs-ofctl mod-port sxx sxx-eth2 down` (prepínač a rozhranie k prepínaču) som vypla dané rozhranie a podľa 3. bodu som kontrolovala zmenu cesty. Zmenená cesta z h1 do h5 je nasledovná:

s00 – s03 – s04 – s08 – s13 – s16

- a) Rovnako ako v predchádzajúcich krokoch som našla cestu z h3 na h7:

s06 – s05 – s09 – s10 – s11 – s15

- b) Zmenila som šírku pásma medzi s09 a s10 na 1000Mbit

- c) Cesta po zmene šírky pásma z h3 na h7 je nasledovná:

s06 – s10 – s9 – s14 – s17 – s18 – s15

Linka, na ktorej bola zmenená šírka pásma je aj naďalej využitá, len sa zmenila celková trasa.

- A) Rovnako ako v predchádzajúcich úlohách som našla cestu medzi h2 a h4

s00 – s03 – s07 – s12

- B) Na rozdiel od predchádzajúceho riešenia som zvolila menšiu šírku pásma na linkách, ktoré sú zaradené do cesty. Zvolila som si linky medzi prepínačmi s00 – s03 a s03 – s07 a dala som im šírku pásma 1Mbit.

- C) Cesta po zmene šírky pásma z h2 na h4 je nasledovná:

s00 – s03 – s07 – s12

Ako je možné vidieť, tak cesta ostáva nezmenená aj po zmene šírky pásma.

- 1) Pomocou príkazov som spustila http server na h1 a poslala naň požiadavku cez h2. Výstup je možné vidieť na obrázku nižšie.

```
root@mininet-vm:~/mininet/custom# python -m SimpleHTTPServer 80 &
[1] 3659
root@mininet-vm:~/mininet/custom# Serving HTTP on 0.0.0.0 port 80 ...
192.168.0.2 - - [15/Nov/2017 15:51:07] "GET / HTTP/1.1" 200 -
192.168.0.2 - - [15/Nov/2017 15:51:29] "GET / HTTP/1.1" 200 -
192.168.0.2 - - [15/Nov/2017 15:52:46] "GET / HTTP/1.1" 200 -
192.168.0.2 - - [15/Nov/2017 15:52:48] "GET / HTTP/1.1" 200 -
```

Obrázok 20. Výstup HTTP servera

5.2. Testovací scénár č.2 (QoS)

Cieľom testovacieho scénára je využiť QoS rad na zistenie podpory QoS v mininete.

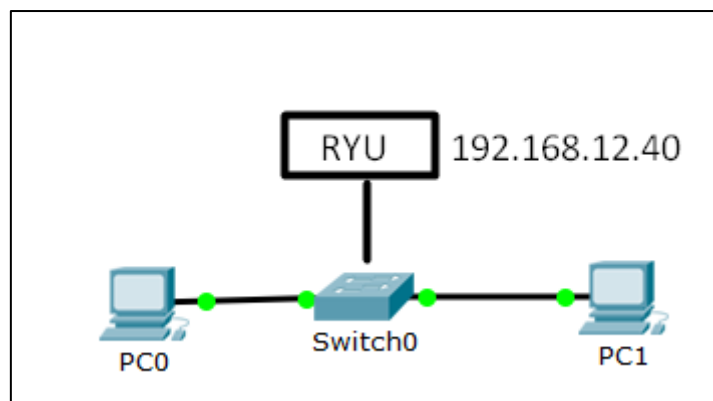
5.2.1. Testovacie prostredie

Operačný systém: Linux

OpenFlow prepínač: Open vSwitch

OpenFlow kontrolér: Ryu kontrolér

5.2.2. Testovacia topológia



Obrázok 21. Testovacia topológia č.2

Switch0 – Open vSwitch

PC0, PC1 – dve koncové zariadenia

5.2.3. Zhodnotenie

- 1) sieť je zhotovená podľa topológie
- 2) ping sa úspešne podaril na všetky zariadenia aj na VLAN

- 3) UDP komunikácia medzi stanicou a koncovým zariadením prebehla úspešne, ale šírka pásma nebola 5Mbit/s alebo 10Mbit/s
- 4) UDP komunikácia prebehla úspešne aj na zariadenia aj VLAN ale bez aplikovania rad
- 5) Nedosiahli sme rovnaké výsledky pretože sa nepodarilo aplikovať rozširovací modul L2QoS.py pre ryu-manager. Všade sme mali základnú 10Mbit/s priepustnosť z nastavenia.

Problémy

- 1) Zlý modul pre ryu-manager, musí sa použiť L2QoS.py pre podporu kontroly OF Switch QoS

5.3. Testovací scenár č.3 (Mininet - wifi)

Cieľom testovacieho scenára je otestovať funkcionality Wi-Fi a ad Hoc sietí v mininete-wifi⁴. Testovací scenár testuje rozdelenie premávky medzi dva body prístupu.

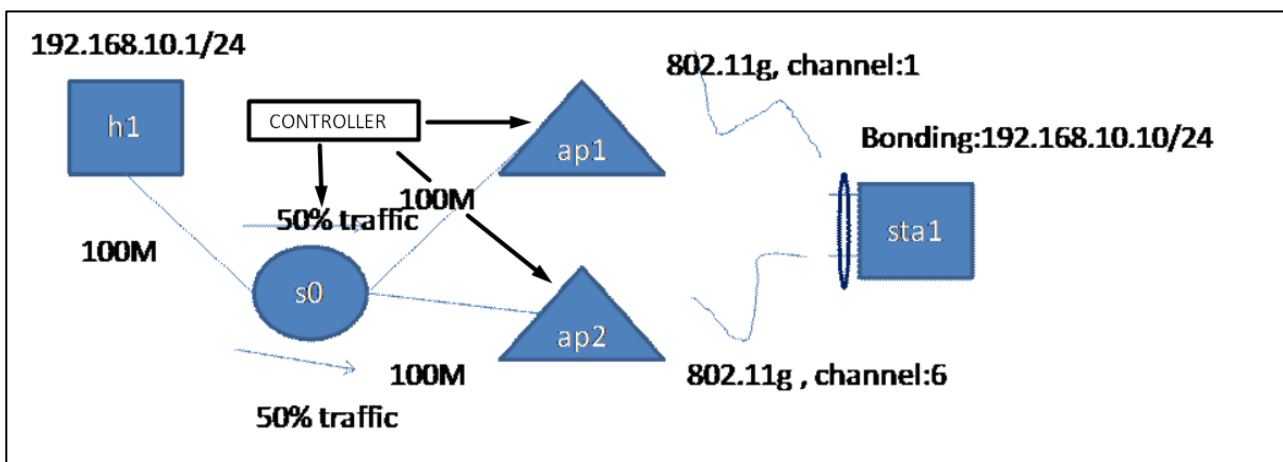
5.3.1. Testovacie prostredie

Operačný systém: Linux

OpenFlow prepínač: Open vSwitch

OpenFlow kontrolér: Ryu kontrolér

5.3.2. Testovacia topológia



Obrázok 22. Testovacia topológia č.3

⁴ <http://sdncentral.ir/wp-content/uploads/2017/07/mininet-wifi-draft-manual.pdf>

s0 – Open vSwitch
h1–koncové zariadenie
controller – ryu kontrolér
ap1, ap2 – body prístupu (access points)
sta1 – stanica

5.3.3. Zhodnotenie

- 1) sieť je zhotovená podľa topológie
- 2) ping sa úspešne podaril na všetky zariadenia
- 3) UDP komunikácia medzi stanicou a koncovým zariadením prebehla neúspešne pri neuvedení manuálnych pravidiel
- 4) UDP komunikácia prebehla úspešne pri použití manuálnych pravidiel
- 5) Dosiahli sa podobné výsledky ako v práci na ktorú konkrétny testovací scenár bol odvodený, prepínač úspešne presmerovával komunikáciu medzi dvomi prístupovými bodmi.

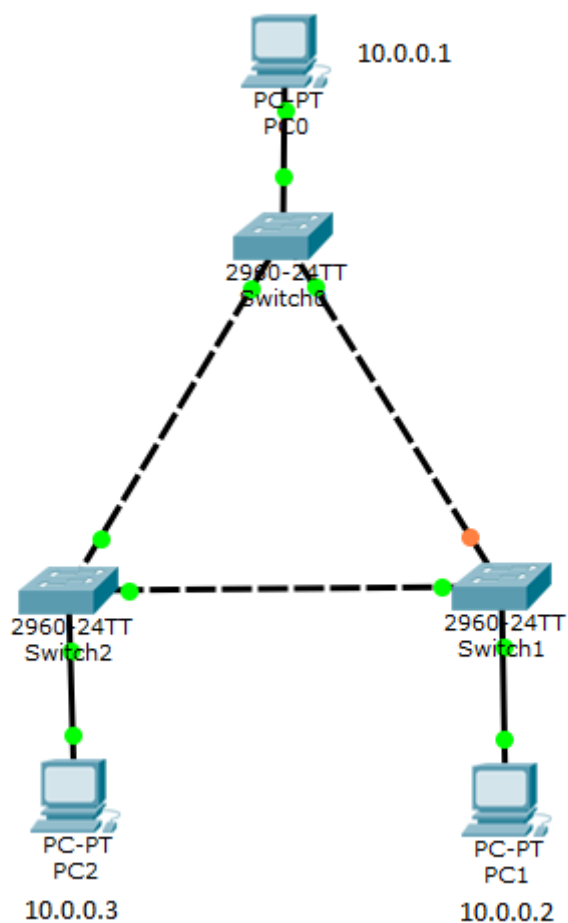
5.4. Testovací scenár č.4

Cieľom testovacieho scenára je overiť fungovanie STP pri vyhodení jednej z liniek.

5.4.1. Testovacie prostredie

Operačný systém: Linux
OpenFlow prepínač: Open vSwitch
OpenFlow kontrolér: Ryu kontrolér

5.4.2. Testovacia topológia



Obrázok 23. Testovacia topológia č.4

5.4.3. Zhodnotenie

Názov testovacieho súboru:

- test3.py

Spúšťacie príkazy:

- `sudo mn --custom test3.py --topo test3 --controller=remote --switch ovs,failMode=standalone,stp=1`
- `ryu-manager /usr/local/lib/python2.7/dist-packages/ryu/app/simple_switch_stp.py`

Testovacie príkazy:

- `sudo ovs-ofctl mod-port s2 s2-eth2 down`
- `sudo ovs-ofctl mod-port s2 s2-eth2 up`

- *sudo ovs-ofctl mod-port s2 s2-eth1 down*
- *sudo ovs-ofctl mod-port s2 s2-eth1 up*
- *links*
- *pingall*

Testovanie zotavenia siete prebehlo úspešne najprv som vypol jednu linku medzi switchom 1 a 2 a potom druhú linku medzi switchom 0 a 2 v oboch prípadoch po naučení všetkých portov bola sieť schopná komunikácie.

6. Literatúra

- [1] SDxCentral. 2017. What is an OpenFlow Controller?
<https://www.sdxcentral.com/sdn/definitions/sdn-controllers/openflow-controller/>
- [2] SHALIMOV, F. et al.: Advanced study of SDN/OpenFlow controllers, Moscow, Russia: Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, 2013 , ISBN 978-1-4503-2641-4
- [3] Thiago Paiva. 2013. OpenFLOW Controllers
https://www.ncc.unesp.br/its/projects/educloud/wiki/OpenFLOW_Collectors/15
- [4] GUDE, N. et al.: NOX: Towards an Operating System for Networks, Yale University, New Haven CT, USA
- [5] David Erickson: The Beacon OpenFlow controller, Stanford University, CA, USA
- [6] KAUR, S., SINGH, J., GHUMMAN, N.S.: Network Programmability Using POX Controller, Ferozepur, India: International Conference on Communication, Computing & Systems, At SBS State Technical Campus , Ferozepur, Punjab , India, 2014
- [7] HARKAL, V.B., DESHMUKH, A.A: Software Defined Networking with Floodlight Controller: International Conference on Internet of Things, Next Generation Networks and Cloud Computing
- [8] SAIKIA, D., MALIK, N.: An Introduction to OpenMUL SDN Suite, 2014
- [9] CAI, Z., COX, A.L., EUGENE, T.S.: Maestro: A System for Scalable OpenFlow Control: Rice University
- [10] Sridhar Rao. 2015. SDN Series Part Eight: Comparison Of Open Source SDN Controllers
<https://thenewstack.io/sdn-series-part-eight-comparison-of-open-source-sdn-controllers/>
- [11] [<https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-59/161-sdn.html>]
- [12] Programming and Communication Channel Issue - [NATARAJAN S., RAMAIAH A., MATHEN M.: "A software defined cloud-gateway automation system using OpenFlow", In: Proc. IEEE 2nd Int. Conf. CloudNet, Nov. 2013, pp. 219-226]
- [13] Potential single point of attack and failure - [BENTON K., CAMP J. L., SMALL CH.: OpenFlow Vulnerability Assessment, Indiana University Bloomington, Indiana, USA, pp. 1-6]
- [14] Mendoca M., Astuto B., Obraczka K., Turletti T.: Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, 2013
- [15] Al-Somaidai M., Yahya E.: Survey of software components to emulate OpenFlow protocol as an SDN implementation, 2014
- [16] Open vswitch and ovs-controller. 2017. <http://openvswitch.org/>

- [17] OpenFlow 1.3 Module for NS-3. 2017. <http://www.lrc.ic.unicamp.br/ofswitch13/>
- [18] Shankdhar, P.: 15 Best Free Packet Crafting Tools. 2017. <http://resources.infosecinstitute.com/15-best-free-packet-crafting-tools/#gref>
- [19] Oficiálna stránka hping. 2017. <http://www.hping.org/>
- [20] Ostinato Traffic generator. 2015. <https://sreeninet.wordpress.com/2015/04/24/ostinato-traffic-generator/>
- [21] Oficiálna stránka dokumentácie Scapy. 2017. <http://www.secdev.org/projects/scapy/>
- [22] Oficiálne úložisko Scapy. 2017. <https://github.com/secdev/scapy>
- [23] Oficiálna stránka PackETH. 2017. <http://packeth.sourceforge.net/packeth/Home.html>
- [24] Python Software Foundation. 2017. <https://pypi.python.org/pypi/ryu/4.18>
- [25] NTT Software Innovation Center. 2013. <https://osrg.github.io/ryu/slides/ONS2013-april-ryu-intro.pdf>
- [26] NTT Software Innovation Center. 2013. <https://osrg.github.io/ryu/slides/LinuxConJapan2013.pdf>
- [27] Tenarys, „QoS“, 2. februára 2016. <https://www.voip-info.org/wiki/view/QoS>
- [28] C. H. Tim Szigeti, Quality of Service Design Overview, 2004
- [29] V. Popeskic, „HOW DOES INTERNET WORK“. 2013. <https://howdoesinternetwork.com/2013/jitter>.
- [30] ONF „Software-Defined Networking (SDN)“. 2017. <https://www.opennetworking.org/sdn-definition/>
- [31] Big Switch Networks. 2017. <https://www.bigswitch.com>
- [32] RFC. 2015. <https://tools.ietf.org/html/rfc7426>

7. Príloha A – Inštalačná príručka

A.1 Inštalácia Ryu

Samotná inštalácia Ryu⁵ je veľmi jednoduchá, vyžaduje iba jeden príkaz:

```
sudo pip install ryu
```

Pre následne spustenie niektorého preddefinovaného komponentu je potrebné vykonať príkaz:

```
ryu run simple_switch.py
```

Navyše, Ryu ponúka interaktívne webové grafické zobrazenie aktuálne spustenej topológie pre jednoduchú vizualizáciu konfigurácie. Tento prvok je dosiahnuteľný aplikovaním prepínača

```
--observe-links
```

K predchádzajúcemu príkazu pre exekúciu programu. Následne je cez webový prehliadač dostupná grafická reprezentácia na danej IP adrese s portom 8080.

⁵ <http://ryu.readthedocs.io/en/latest/index.html> - Dokumentácia a postup pre modifikáciu Ryu

8. Príloha B – Používateľská príručka

B.1 Mininet

Diagnostické príkazy

- **help**
 - zobrazenie dostupných príkazov
- **nodes**
 - zobrazenie uzlov zapojených v topológii
 - c0, h1, h2, s1
- **net**
 - zobrazenie informácie o linkách medzi uzlami
 - h1-eth0:s1-eth
 - s1-eth2:h2-eth0
- **dump**
 - zobrazenie informácie o všetkých zariadeniach v topológii
- **h1 ifconfig -a**
 - zobrazenie podrobnejších informácií o zariadení
- **h1 arp**
 - zobrazenie ARP tabuľky pre špecifikované zariadenie
- **h1 route**
 - zobrazenie smerovania pre špecifikované zariadenie
- **mn --v output**
 - zrušenie výpisov do konzoly pri vytváraní topológie
- **mn --v debug**
 - vynútenie všetkých výpisov do konzoly pri vytváraní topológie

Práca s topológiou

- **mn --topo minimal**
 - základná topológia
 - jeden kontrolér, jeden prepínač, dve koncové zariadenia
- **mn --topo linear, #**
 - sériové zapojenie prepínačov s jedným koncovým zariadením na prepínač
- **mn --topo single, #**
 - topológia s # koncovými zariadeniami na jeden prepínač
- **mn --switch ovs --controller ref --topo tree,depth=2,fanout=8 --test pingall**
 - Vytvorí stromovú topológiu kde na každý uzol napojí 8 hostov, dohrom. 64
 - Použije Open vSwitch a OpenSwitch/Stanford reference controller
 - Následne spustí ping medzi všetkými zariadeniami
- **link s1 h1 down / up**
 - zmena stavu linky medzi dvoma zariadeniami

- **mn --link tc, bw=10, delay=10ms**
 - nastavenie šírky pásma a oneskorenia medzi linkami
- **mn --mac**
 - vynútenie nastavenia jednoduchých MAC adries pre rozhrania
- **mn --c**
 - odstránenie topológie
- **h1 python -m SimpleHTTPServer 80 &**
 - spustenie http serveru na špecifikovanom zariadení
 - (h1 python -m SimpleHTTPServer 80 >& /tmp/http.log &)
- **h2 wget -O - h1**
 - vyžiadanie HTTP spojenia medzi špecifikovanými zariadeniami

Overenie funkčnosti topológie

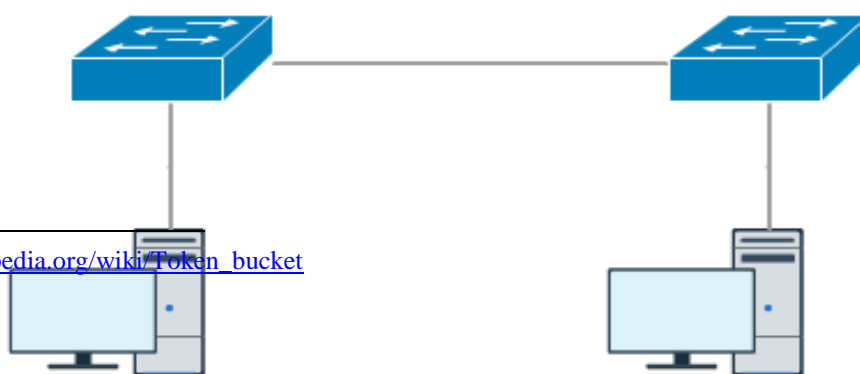
- **h1 ps**
 - zobrazenie procesov celého zapojenia
- **h1 ping --c # h2**
 - ping medzi koncovými zariadeniami # krát
- **pingall**
 - ping na všetkých spojeniach
- **mn --test pingpair**
 - automatické vytvorenie topológie, overenie pomocou ping na všetkých linkách a zrušenie topológie

Vytvorenie vlastnej topológie

V aplikácii Mininet je možné vytvárať topológie podľa vlastných potrieb s požadovanými parametrami. Syntax vychádza z jazyka Python a je ľahko pochopiteľná a jednoznačná.

Nasledujúci kód vytvorí jednoduchú topológiu s dvoma prepínačmi, ktoré sú spoločne prepojené a každý má jedno koncové zariadenia. Taktiež linka medzi pravým prepínačom a jeho pripojeným zariadením je upravená nasledovne:

- šírka pásma 10 Mbps
- oneskorenie 5 ms (us, s)
- strata 2%
- maximálna dĺžka radu 1000 paketov
- s využitím princípu *Hierarchical Token Bucket*⁶



⁶ https://en.wikipedia.org/wiki/Token_bucket

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, rightSwitch )
        self.addLink( rightSwitch, rightHost, bw=10, delay='5ms',
            loss=2, max_queue_size=1000, use_htb=True )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

9. Príloha C – Priebeh testovania

C.1 Testovací scenár č.1

1. over, či sú všetky linky zapojené tak, ako je potrebné //net
2. pingom over dostupnosť všetkých koncových zariadení //pingall
3. nájdi cestu z PC1 do PC5 //tracert -n IPaddress
4. zhod' linku na ceste, získanú z bodu 3 // ifconfig sX-ethX down
5. zopakuj krok 3 a over, či linka bola zmenená a sieť naďalej funguje
 - a) nájdi cestu z PC3 do PC7
 - b) zmeň šírku pásma na trase, získanej z bodu a)
 - c) zopakuj krok a) a over, či bola cesta zmenená

- A. nájdi cestu z PC2 na PC4
- B. na ceste získanej z bodu A. zmeň šírku pásma na niektorej linke
- C. zopakuj krok A. a zisti, či bola cesta zmenená

- 1) vytvor http server //h1 python -m SimpleHTTPServer 80 &;
- 2) pošli požiadavku na h2 //h2 wget -O - h1
- 3) vypni server //h1 kill %python

C.2 Testovací scenár č.2

- 1) Priradiť PC0 VLAN port a IP adresu (pri port 100 10.0.0.2/24, adresa 1.0.0.2/24)
- 2) Priradiť PC1 VLAN port a IP adresu (pri port 100 10.0.0.3/24, adresa 1.0.0.3/24)
- 3) Nastav Open vSwitch (Switch0) (priradiť porty s koncovými zariadeniami)
- 4) Vytvor Queue ⁷vo Switch0
- 5) Inštalácia toku pre PCP (Priority Code Point) do radu mapovania je zabezpečená RYU kontrolérom
- 6) Ping na PC1 (1.0.0.3) z PC0 – mal by fungovať
- 7) Ping na PC0 (1.0.0.2) z PC1 – mal by fungovať
- 8) Overenie toku tabuľky tokov pridaním VLAN tagu (PCP = 3)
- 9) Ping na PC1 (10.0.0.3) z PC0 – funguje
- 10) Ping na PC0 (10.0.0.2) z PC1 – funguje
- 11) Skontrolovanie šírka pásma pre tok bez VLAN tagu
- 12) Spustenie iperf servera na PC0 (iperf -s -p 8011 -u -f m -reportstyle C -i 1 //udp server)
- 13) Spustie iperf klienta na PC1 (iperf -c 1.0.0.3 -p 8011 -i 1 -f m -t 120 -u -b 10000000)
- 14) Overenie, že šírka pásma na PC1 je 10Mbps/s
- 15) Skontrolovanie šírky pásma pre tok s VLAN tagom (PCP = 3)
- 16) Spustenie iperf servera na PC0 (iperf -s -p 8011 -u -f m -reportstyle C -i 1 //udp server)
- 17) Spustie iperf klienta na PC1 (iperf -c 10.0.0.3 -p 8011 -i 1 -f m -t 120 -u -b 10000000)
- 18) Overenie, že výstup na PC1 je 5Mbps/s

⁷ <https://www.networkworld.com/article/2234323/cisco-subnet/quality-of-service--queuing.html>

C.3 Testovací scenár č.3

- 1) Nastavíme topológiu podľa obrázka 2.
- 2) Na s0 nastavíme, aby preniesol 50 % premávky cez ap1 a zvyšných 50 % cez ap2 do sta1
- 3) Otestujeme pomocou príkazov
- 4) Na sta1: `iperf -s -u -i 1`
- 5) Na h1: `iperf -c 192.168.10.10 -u -b 100M -t 5`
- 6) V prípade problémov konfigurácie kontroléra, je možné danú topológiu zostrojiť a otestovať aj bez neho.

C.4 Testovací scenár č.4

- 1) definuj kontrolér a sieť `//netMininet(controller=RemoteController)`
- 2) vytvor kontrolér s portom 6633 `//net.addController(name, port)`
- 3) vytvor 3 prepínače `//net.addSwitch(name)`
- 4) vytvor tri koncové zariadenia `//net.addHost(name,ip)`
- 5) vytvor linky podľa topológie `//net.addLink(src, dts)`
- 6) spusti sieť `//net.build(),xy.start(controller),et.startTerms()`
- 7) zapni CLI `//CLI(net)`
- 8) nastav OpenFlow verziu pre prepínače `//ovs-vsctl set Bridge sxp protocols=OpenFlow13`
- 9) na uzle root (Node: c0 (root)) spusti skript pre STP
- 10) `ryu-manager ryu.app.simple_switch_stp_13`
- 11) pre všetky uzly (nodes) nastav filtrovanie TCP/IP paketov
- 12) `tcpdump -i sx-eth2 arp`
- 13) pingom z host1 do host2 over, že paket nie loop-ovaný
- 14) zhod' eth linku s2 `// # ifconfig s2-eth2 down`
- 15) linka, ktorá bola v stave blocked, by sa mala prepnúť do stavu forwarding