Dynamic Traffic Diversion in SDN: Testbed vs Mininet

Bc. Lukáš Mastiľak, Bc. Ján Pánis, Bc. Andrej Vaculčiak

***Abstrakt*—Základom vývoja v ktorejkoľvek oblasti je testovanie. V drvivej väčšine prípadov potrebujeme testovať počas celého procesu vývoja, avšak nie vždy máme k dispozícii koncové zariadenia, resp. hardvér, na ktorý je daná aplikácia vyvíjaná. Preto vznikli rôzne emulátory, ktoré viac, či menej dôveryhodne emulujú reálne prostredie a tým uľahčujú proces vývoja a testovania riešenia.**

Obsahom nasledujúceho článku je analýza pôvodného článku [1], ktorý sa zaoberá overením relevantnosti výsledkov poskytujúcich emulátorom SDN sietí s názvom Mininet voči výsledkom získaným reálnym zapojením SDN siete. V úvodnej časti predstavuje problematiku samotných SDN sietí a prečo je dobré sa nimi zaoberať. Okrem toho opisuje pôvodné riešenie podľa článku [1], spôsob testovania a vyhodnotenie testov, ktoré realizovali.

Na ich základe sme následne navrhli a tak isto testovali prostredie Mininet a modifikovanú reálnu fyzickú topológiu založenú na zariadeniach Soekris net6501. Výsledky testov sme podľa možností porovnali s riešením v pôvodnom článku [1].

***Index Terms*—Mininet, Reálne prostredie, DTD, Dynamic Traffic Diversion algoritmus**

I. Úvod

Pre vývoj v oblasti sieťových zariadení je pomerne často využívaný emulovaný systém. Má množstvo výhod, no tak isto aj svoje nevýhody. Pre SDN siete platí podobné pravidlo. Avšak, naskytá sa otázka, či je emulovaný systém dôveryhodným zdrojom výsledkov.

Overenie tejto problematiky bolo témou článku, ktorý sme si vybrali. Porovnáva reálne prostredie, tvorené fyzickými prepínačmi s podporou SDN a emulátorom Mininet.

II. Analýza článku

Táto časť dokumentu sa venuje opisu pôvodného riešenia podľa referenčného článku, spolu s výsledkami testovania.

A. Software Defined Networks

Posledných niekoľko rokov sa do popredia v oblasti sietí dostali tzv. SDN, čiže softvérovo definované siete. Základnou myšlienkou týchto sietí je oddelenie riadiacej časti od dátovej. Každá dnes bežná sieť je zložená z komponentov, ktoré obsahujú hardvérovú časť, zabezpečujúcu smerovanie a prepínanie, a softvérovú časť, ktorá rieši spracovanie požiadaviek, prípadné výpočty pre smerovacie protokoly a pod.

Problém ale je, napr. v pomerne zložitom spôsobe zabezpečenia kompatibility, či potrebe nastavovania každého prvku siete samostatne ale aj závislosti na podpore od výrobcu [1, 2].

Preto sa prišlo s myšlienkou oddelenia kontrolnej časti sieťových prvkov a vytvoriť jeden centrálny prvok (tzv. controller), ktorý zabezpečí kontrolu nad sieťou. Výhodou takéhoto prístupu je, o. i. centralizovaná konfigurácia siete, možnosť videnia celej topológie z pohľadu controllera, čím sa napríklad odstraňuje potreba smerovacích protokolov prítomných v každom zo sieťových zariadení a zbytočne dlhá konvergencia siete pri výpadku. Ako ďalšie pozitívum sa vníma aj oveľa zjednodušená možnosť aplikácie tzv. traffic engineering-u [2].

B. Topológia

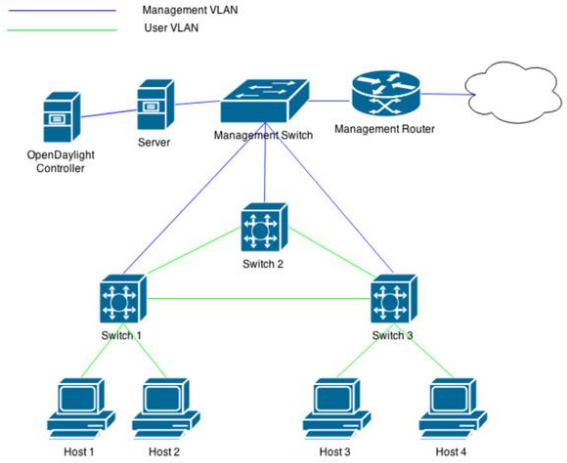
Základnú časť topológie siete na Obr. 1 tvoria tri SDN prepínače, ktoré sú navzájom prepojené kvôli redundantným cestám. Prepínače sú označené číslami 1 až 3. K prepínačom 1 a 3 sú pripojené dve koncové zariadenia. Daný návrh topológie ponúka pri komunikácií medzi koncovými zariadeniami viacero ciest, ktorými môže komunikácia prebiehať. Jedna z nich slúži ako primárna cesta a v prípade, že na nej vznikne zahltenie, premávka sa presmeruje cez záložnú cestu. Takto je možné udržať jitter a straty paketov na minime. V návrhu topológie sa počíta aj s ošetrením proti možnému vzniku slučiek.

Ďalším prvkom topológie je controller OpenDaylight, ktorý beží na virtuálnom serveri s OS Ubuntu. Výber daného controllera bol ovplyvnení širokou podporou Java na rôznych platformách. OpenDaylight bol použitý tak pre reálnu implementáciu, ako aj pre prostredie Mininet [1].

Pre realizáciu fyzickej topológie boli vybrané prepínače Cisco Catalyst 3650, na ktorých bežala trial verzia IOS-XE s podporou pre OpenFlow. Autori počítali s podporou verzie OpenFlow 1.3, ale komunikácia medzi prepínačmi a controllerom nefungovala, preto sa rozhodli pre vyskúšanie verzie 1.0, kde komunikácia už bola funkčná. Všetky porty na prepínačoch boli nastavené na 100 Mbit/s. Koncové zariadenia disponovali OS Lubuntu a na každom z nich bol nainštalovaný nástroj na meranie výkonu siete Iperf [1, 3, 4].

Tento článok vznikol ako výsledok semestrálneho projektu z predmetu Architektúry komunikačných systémov/Distribúcia obsahu v Internete.

Autori sú študenti študijneho programu Softvérové inžinierstvo v študijnom odbore Softvérové inžinierstvo (akad. r. 2017/2018)
E-mail: xmastilak@stuba.sk, xpanis@stuba.sk, xvaculciak@stuba.sk



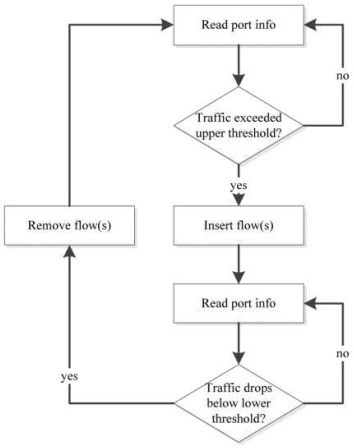
Obr. 1 – Návrh topológie [1]

C. *Algoritmus DTD*

Algoritmus DTD (Dynamic Traffic Diversion) bol vytvorený pre testovacie účely. Je pomocou neho možné dynamicky meniť tok premávky, za účelom zníženia stratovosti paketov a jitter-u [1].

Strata paketov je spôsobená zlyhaním prenosu paketov, ktoré prichádzajú do cieľa, zatiaľ čo jitter je kolísanie veľkosti oneskorenia paketov počas prenosu sieťou [1].

Úlohou algoritmu je v pravidelných intervaloch vyhodnocovať vyťaženosť portov na smerovačoch a v prípade zahltenia (resp. prekročenia stanovenej hranice vyťaženia linky), odľahčiť tok dát použitím záložnej linky. Ak v nejakom okamihu klesne úroveň vyťaženosti opäť na prijateľnú hodnotu, záložná linka sa prestane využívať a premávka bude posielaná cez prioritnú linku. Popis algoritmu je tiež možné vidieť v aktivity diagrame na Obr. 2 [1].



Obr. 2 – Aktivity diagram algoritmu DTD

Hranice vyťaženosti linky boli vyčíslené na hodnoty:

- Horná hranica - 90% kapacity linky (v prípade, že ide o 100Mbit/s pri prekročení 90Mbit/s, nastane presmerovanie toku)
- Dolná hranica - 70% kapacity linky [1].

Na vytvorenie aplikácie sa rozhodli využiť programovací jazyk Python z dôvodu, že pri základnom testovaní využívali klasický program príkazového riadku - cURL, ktorý je možné využívať aj v Python-e. Ten bol využitý aj na získavanie informácií o portoch dostupných pomocou OpenDaylight Northbound REST API [1].

Pre otestovanie algoritmu zostrojili testovaciu topológiu skladajúcu sa z 3 hlavných prepínačov, prepojených každý s každým (jedna cesta je primárna, druhá sekundárna). Viac o topológií je písané v kapitole Topológia [1].

Na testovanie rýchlosti, straty paketov a jitter-u využili program Iperf, ktorý je zároveň generátorom paketov a tokov, ako aj nástrojom pre rôzne merania v sieti. Daný program sa využíva na ladenie výkonu v sieťach a medzi jeho hlavné výhody patria:

- Schopnosť fungovania na rôznych platformách (Windows, Unix, Linux)
- Otvorený zdrojový kód napísaný v jazyku C
- Umožňuje jednosmerné, ale aj obojsmerné merania
- Dáta môžu byť vysielané protokolom UDP ale aj TCP s nastaviteľnými veľkosťami okien [1, 3, 4]

D. *Testovanie*

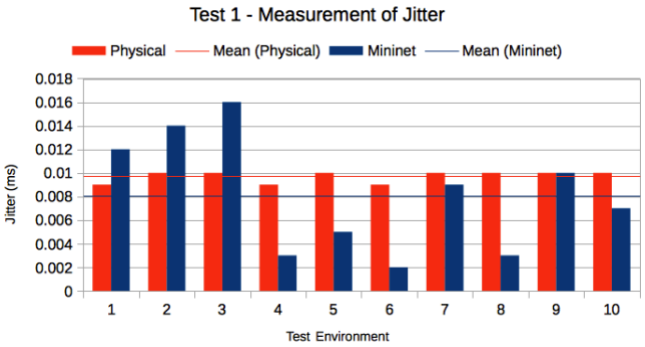
Obsahom testovania bolo overenie hypotézy o dôveryhodnosti výsledkov, ktoré ponúka Mininet oproti reálnemu prostrediu, tvorenému Cisco prepínačmi s podporou SDN. Prenosová rýchlosť liniek bola nastavená na 100Mbit/s [1].

Túto hypotézu overovali pomocou implementácie DTD algoritmu a sledovaní správania sa siete v rôznych scenároch. Testované boli 3 scenáre:

1. Základný test
2. Výkonnostný test bez DTD
3. Výkonnostný test s použitím DTD [1]

1) *Základný test*

Úlohou základného testu je zistiť počiatočné podmienky a vlastnosti (jitter, stratovosť paketov a ich oneskorenie) danej topológie. Spočíva v UDP komunikácií uzlov H1 a H3. Uzol H1 posiela 600MB dát rýchlosťou 50Mbit/s a nakoľko tam nie je žiadna iná premávka, nedochádza k strate paketov a jitter je spôsobený len oneskorením na linkách [1].

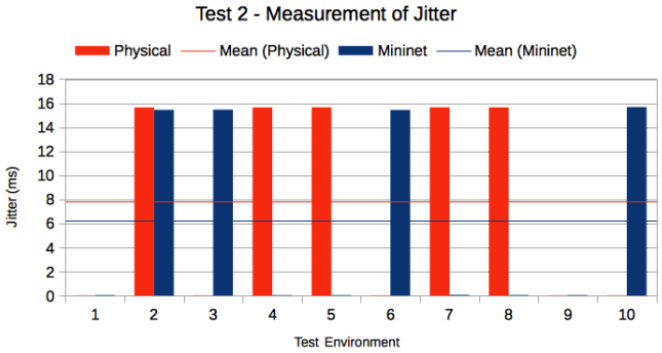


Obr. 3 – Porovnanie hodnôt jitter-u pre obe testované prostredia (scenár 1) [1]

2) *Výkonnostný test bez DTD*

Hlavným cieľom tohto scenára bolo určiť správanie sa siete počas zahltenia. Uzly H1 a H3 komunikujú rovnako ako v prípade 1. testu. Okrem nich však do siete pribudla komunikácia uzlov H2 a H4, ktoré si posielajú veľké množstvo

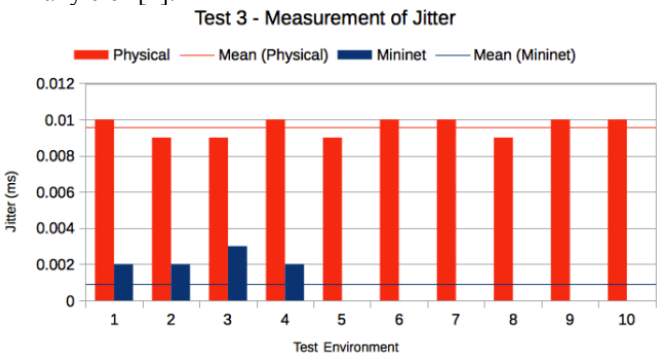
UDP paketov rýchlosťou 95Mbit/s. Táto skutočnosť zapríčiní zahltenie linky medzi prepínačmi a teda zvýši sa stratovosť paketov a aj jitter, čo je predpokladaný jav. Začnú sa strácať pakety, pričom táto stratovosť dosiahla v priemere 50% pre reálne prostredie a 34% v Mininete [1].



Obr. 4 - Porovnanie hodnôt jitter-u pre obe testované prostredia (scenár 2) [1]

3) Výkonnostný test s použitím DTD

Cieľom tohto testu bolo overiť hypotézu, či aplikácia za použitia DTD dokáže znížiť jitter a zvýšiť celkovú doručiteľnsoť paketov (packet loss). Scenár je v podstate identický s predchádzajúcim, no pribudla v ňom situácia, v ktorej, keď dôjde k zahlteniu linky na viac ako 90%, presmeruje sa premávka z H1 do H3 na záložnú linku, čím sa má dosiahnuť spomínaný cieľ [1].



Obr. 5 - Porovnanie hodnôt jitter-u pre obe testované prostredia (scenár 3) [1]

Ako už bolo spomenuté, maximálna hodnota využitia linky, ktorá sa považuje za kritickú, je 90%, čiže 90Mbit/s. Avšak, v prípade, že využitie linky kleslo, tým pádom už zahltenie na prioritnej ceste nemáme, premávka sa zo záložnej linky presmeruje opäť na prioritnú linku. Táto situácia nastane, ak je využitie danej linky pod 70%, a teda 70Mbit/s [1].

E. Vyhodnotenie testovania

Ako môžeme vidieť na grafoch zobrazujúcich výsledky meraní, autori prišli k záveru, že ich hypotéza o dôveryhodnosti výsledkov, ktoré poskytuje Mininet oproti reálnemu prostrediu a faktu, že nedochádzalo k stratám paketov pri aplikovaní DTD, bola správna.

Avšak, pri testovaní 3. scenára sa objavila veľmi malá hodnota jitteru v Mininet prostredí, čo sa ale dá odôvodniť faktom, že Mininet je integrovaný na jednom stroji. Okrem toho neaplikovali do Mininetu oneskorenie na linkách, čo koniec koncov tak isto ovplyvnilo výsledky meraní a rozdiel v priemerných hodnotách [1].

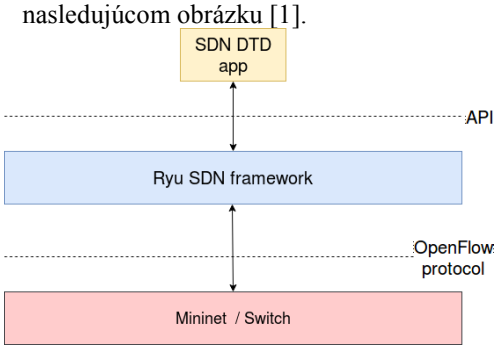
III. NÁVRH PROJEKTU

Náš projekt sa skladá z dvoch hlavných častí, ktoré overujú referenčný článok (otestovanie algoritmu DTD, porovnanie oneskorení, rýchlosti a jitter-u):

1. v emulátore Mininet,
2. a na reálnych zariadeniach Soekris net6501 [1].

Architektúra SDN prostredia sa skladá z nasledujúcich prvkov a je pre oba prípady totožná.

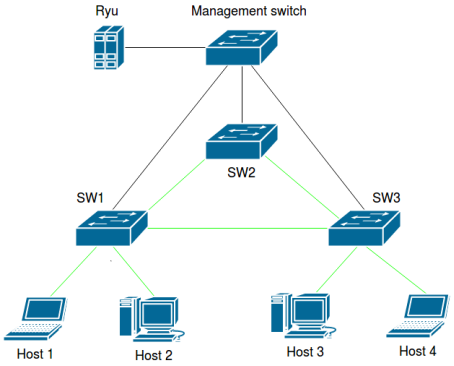
- RYU SDN controller, ktorého úlohou je poskytovať medzivrstvu medzi prepínačmi (či už reálnymi, alebo emulovanými v Mininet-e ktoré podporujú OpenFlow) a externou aplikáciou. Okrem iného korektne riadi tok dát v sieti, ktorý môže byť dynamicky upravený pomocou aplikácie.
- Aplikácia s RYU controllerom komunikuje pomocou REST API a následne cez tohto prostredníka sa dostávajú riadiace informácie do prepínačov. Celú situáciu je taktiež možno vidieť na nasledujúcom obrázku [1].



Obr. 6 - Návrh architektúry

A. Mininet

V návrhu našej topológie sme sa rozhodli trochu upraviť pôvodnú topológiu tým, že niektoré časti sme sa rozhodli vynechať. Konfigurácia siete bude na základe controllera. Chceme minimalizovať akúkoľvek konfiguráciu na prepínačoch. Určenie primárnej a v prípade potreby sekundárnej cesty sa bude riešiť cez controller podobne ako v pôvodnom riešení.



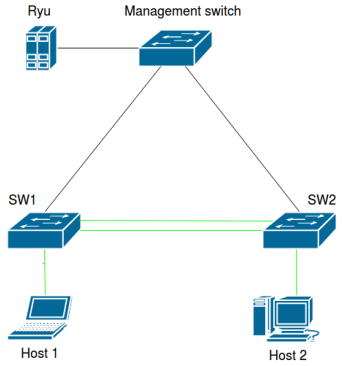
Obr. 7 - Návrh topológie pre Mininet

B. *Reálne prostredie*

Na testovanie SDN siete v reálnom prostredí používame zariadenie Soekris net6501, na ktorom je OS Debian. Architektúra je podobná s Mininet architektúrou. Rozdiel je v použití fyzického SDN prepínača v spolupráci s úplne bežným prepínačom, namiesto Mininet emulátora. V prepínači je na OS Debian spustený proces Open vSwitch, ktorý podporuje OpenFlow. Daný prepínač komunikuje s controllerom, ktorý je implementovaný pomocou RYU a nad ním je aplikácia, ktorá implementuje DTD algoritmus. Aplikácia pomocou RYU API dáva inštrukcie prepínaču, aby sa vytvorila cesta pre konkrétne pakety záložnou cestou (v prípade potreby).

Čo sa ale týka zapojenia a samotnej topológie, bolo potrebné pristúpiť k zmenám, nakoľko nemáme k dispozícií taký počet SDN prepínačov a ani portov na prepínačoch.

Nakoľko hardvérový SDN prepínač disponuje len 4 portami, z toho 1 je použitý pre komunikáciu zariadenia s controllerom, čiže prakticky sú k dispozícií len 3 porty. Z tohto dôvodu sa zmenšil počet hostov zo 4 na 2.



Obr. 8 - Návrh topológie pre reálne prostredie

C. *Implementácia navrhovaného riešenia*

Obsahom tejto časti je opis konkrétnej implementácie DTD algoritmu spolu s potrebnými skriptmi pre prípravu jednotlivých prostredí na testovanie.

1) *Mininet*

Rovnako ako autori pôvodnej práce, tak aj my sme použili virtuálny stroj Mininet. Topológiu sme zachovali rovnakú. Konfiguráciu topológie sme ešte vylepšili o to, že sme definovali pre jednotlivých hostov pevné MAC adresy v rozsahu 00:00:00:01 až 00:00:00:04, kde koncové číslo označuje číslo hosta.

Ďalej sme každej linke určili čísla portov na zariadeniach, ktoré spájajú. Tak sme zabezpečili, že naša aplikácia nad Ryu bude stále sledovať správny port. Šírku liniek sme zachovali na pôvodnej hodnote 100 Mbit/s.

2) *Dynamic Traffic Diversion aplikácia*

Nad Ryu sme postavili aplikáciu s názvom scriptu dtd_app.py. V aplikácii sme implementovali DTD algoritmus, ktorý bol navrhnutý pôvodnými autormi. Ak primárna trasa bola zahltená na 90% zo svojej kapacity, tak sa pre hosta H1 zmenila trasa do H3 cez záložnú cestu a takisto v opačnom smere. Keď vyťaženosť primárnej linky klesla pod 70% svojej kapacity, tak sa trasa pre H1 a H3 vrátila na primárnu cestu a záložná cesta bola zrušená.

Ďalej naša aplikácia nastavuje základnú konfiguráciu prepínačov. Vytvorí a pošle konfiguráciu na daný prepínač pre vytvorenie nového flowu. Taktiež zabezpečuje modifikáciu existujúcich flowov pri zmene ciest. Ak to zhrnieme, tak naša aplikácia sa postará aj o nastavenie pravidiel na prepínačoch pre flowy. Nie je potrebné nič manuálne konfigurovať.

Nakoniec sme pridali aj štatistický výpis do terminálu pre port sledovaný na primárnej a záložnej ceste. Uvádzame koľko bajtov bolo na danom porte prenesených a aktuálnu vyťaženosť na porte.

3) *Reálne prostredie (Soekris net6501)*

Aby sme mohli pracovať so zariadením pomocou controllera, ktorý komunikuje prostredníctvom protokolu OpenFlow, je potrebné zariadenie na to pripraviť. Nainštalovanie aplikácie openvswitch-switch spolu s potrebným nastavením rozhraní je realizované pomocou skriptu init_setup.bash, ktorý okrem toho vykoná aj nastavenie používanej verzie OpenFlow na verzie 1.0 a 1.3.

Pre jednoduchšiu prácu s portom využívaným na manažment (pripojený na controller) bol vytvorený skript (switch_setup.py), ktorý pomocou prepínačov umožňuje zmeniť pridelenú adresu/masku/predvolenú bránu pomocou DHCP na statickú a tak isto umožňuje zmeniť aj adresu a port controllera.

Posledný skript je upravená verzia skriptu DTD algoritmu pre mininet topológiu. Úprava bola nevyhnutná z dôvodu rozdielnosti topológií a počte OF zariadení.

D. *Zhodnotenie a porovnanie emulovaných a reálnych výsledkov*

Výsledky práce v prvom rade dospeli k tomu, že navrhovaný algoritmus DTD je vhodné použiť na minimalizáciu jitteru a stratovosti paketov. Ďalej je tento algoritmus vhodný na riešenie zahltenia na primárnej linke. Takto je možné dosiahnuť, aby premávka s vysokou prioritou nebola blokovaná premávkou s nižšou prioritou na primárnej linke.

Druhý výsledok práce je porovnanie výsledkov nášho testovania v prostredí Mininet a testovania autorov pôvodného článku. Dosiahnuté výsledky sú porovnateľné, s malými rozdielmi [1].

Tretím výsledkom je zhotovenie HW prostredia, do ktorého sme aplikovali algoritmus DTD. Merali sme rovnaké veličiny ako sú jitter a stratovosť paketov, avšak vzhľadom na naše / pôvodné merania v prostredí Mininet (pri ktorom sme vychádzali z inej topológie a iných rýchlosti liniek, vzhľadom na HW, ktorým sme disponovali - 1 prepínač Soekris net6501 s obmedzenou rýchlosťou liniek na 10Mbit/s), nie je možné tieto merania až tak priamo porovnávať.

1) *Výsledky našich meraní*

Ako bolo spomenuté v kapitole testovanie, prevzali sme scenáre z pôvodného článku.

K úpravám dochádza len v prípade reálneho prostredia, nakoľko nebolo možné zapojiť toľko zariadení a využiť všade 100Mbit linky. Bolo preto potrebné pristúpiť k zníženiu prenosovej rýchlosti medzi prepínačmi a to na 10Mbit (iné zmenšenie zariadenie Soekris neposkytuje) a teda všetky parametre meraní boli v prípade reálneho zapojenia a testovania 10-násobne zmenšené, aby sa zachoval pomer v súlade z mininet testovaním.

Testovania vždy pozostávajú z 2 častí a to testovania pre prostredie Mininet a pre HW (ich opis už špeciálne rozoberať nebudeme, pretože vykonávame rovnaké testy ako autori článku - kapitola Analýza článku, Testovanie)

1. Základný test
2. Výkonnostný test bez DTD
3. Výkonnostný test s použitím DTD [1]

Výsledky sme, podobne ako autori pôvodného článku, zhrnuli do grafov nachádzajúcich sa v nasledujúcich kapitolách.

a) Testovanie v prostredí Mininet

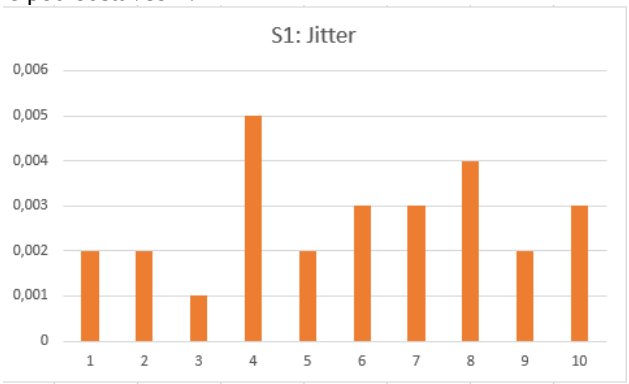
Nasleduje opis výsledkov testov v Mininet emulátore.

(1) Základný test

Čo sa týka stratovosti paketov v našom Mininet meraní, sme dospeli k takmer identickému záveru ako autori článku. Spriemerované hodnoty 10-tich meraní nám vykazujú hodnotu percentuálnej stratovosti paketov len 0,00079%. Je možné, že autorovi stačilo pracovať s menšou presnosťou, pretože on uvádza stratovosť paketov o hodnote 0,0%. V tomto prípade sme namerali teda rovnaké výsledky ako autori. Viď graf percentuálnej stratovosti paketov pod odstavcom.

Obr. 9 - Stratovosť paketov v prostredí mininet v základom teste

Hodnoty jitter-u nám vyšli tiež o čosi lepšie ako autorom, pretože priemerná hodnota jitter-u po 10-tich meraniach u nás nadobudla hodnotu 0,0027 ms. V autorovom meraní získali priemernú hodnotu 0,0081. Namerané hodnoty možno vidieť v grafe pod odstavcom.



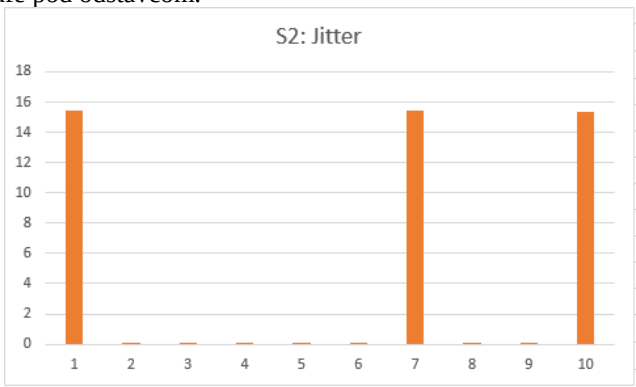
Obr. 10 - Jitter v prostredí mininet v základom teste

(2) Výkonnostný test bez DTD

Výkonnostný test, v ktorom sme zaťažovali primárnu cestu tokom UDP dát o rýchlosti 95 Mbit/s dopadol v našom prípade nasledovne. Priemerná percentuálna hodnota stratovosti paketov sa pohybovala na úrovni 40,2728%. Pre porovnanie autori namerali lepšiu hodnotu a to 34%. Ich hodnoty sú teda o cca 6% lepšie. Nami namerané hodnoty možno vidieť pod odstavcom.

Obr. 11 - Stratovosť paketov v prostredí mininet vo výkonnostnom teste

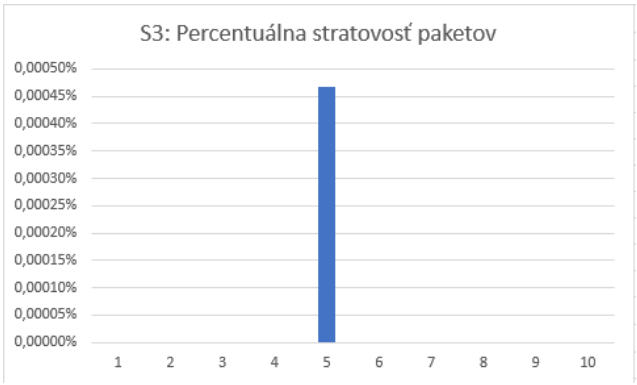
Naopak v ich testovaní dosiahli horšie hodnoty jitter-u a ich priemerná hodnota sa zastavila na úrovni 6,2207ms. Nám sa podarila namerať priemerná hodnota iba 4,6393ms čo je o 1,5814ms lepšie. Namerané hodnoty je taktiež možno vidieť v grafe pod odstavcom.



Obr. 12 - Jitter v prostredí mininet vo výkonnostnom teste

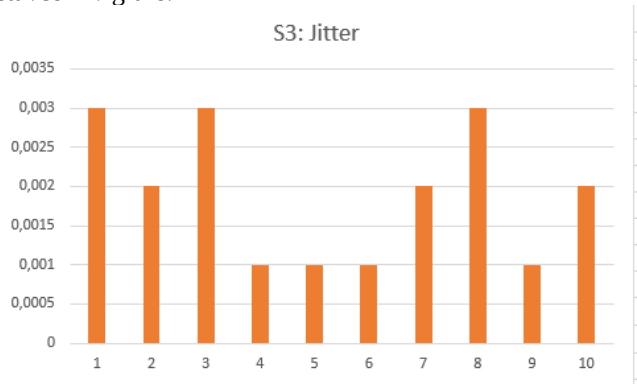
(3) Výkonnostný test s použitím DTD

Rovnako ako v prvom teste autori uvádzajú hodnotu percentuálnej stratovosti paketov o hodnote 0,0%. Nám sa však v prostredí Mininet podarilo namerať priemernú percentuálnu hodnotu stratovosti paketov 0,00005%. Táto hodnota je veľmi dobrá, pretože vo všetkých testoch z odoslaných 4279900 paketov sa stratili len 2. Je zaujímavé, že v tomto prípade nám vyšla nižšia stratovosť ako v referenčnom (aj keď iba o málo). Namerané hodnoty možno vidieť v grafe pod odstavcom.



Obr. 13 - Stratovosť paketov v prostredí mininet vo výkonnostnom teste s použitím DTD

V meraní hodnôt jitter-u sa podarilo autorom nameraťo trochu lepšie hodnoty ako nám. Ich priemerná hodnota činila 0,001 ms a naša 0,0019, čo je o 9 desaťtisícin horšia hodnota. Každopádne ak sa na všetky testy pozeráme ako na celok, vyšli nám, v podstate, rovnaké hodnoty, a teda môžeme pôvodné merania len potvrdiť a považovať ich za relevantné. Naše merania jitter-u z posledného testu je možno vidieť pod odstavcom v grafe.



Obr. 14 - Jitter v prostredí mininet vo výkonnostnom teste s použitím DTD

b) Testovanie v reálnom prostredí

Ako už bolo spomenuté, keďže sme nemali rovnaký počet SDN prepínačov, museli sme si topológiu zjednodušiť (viď obrázok 8 - Návrh topológie pre reálne prostredie). Okrem iného sme museli medzi prepínačmi znížiť kapacitu liniek na 10 Mbit/s. Problém bol taký, že prepínač mal iba 4 porty. Prvý rezervovaný manažmentom, 2 porty pre vytvorenie redundantného prepojenia medzi prepínačmi a jeden pre pripojenie aspoň jedného hosta. Aby sme dokázali využiť kapacitu oboch liniek medzi prepínačmi, bolo nutné ich rýchlosť limitovať aspoň na polovicu (teda zo 100Mbit/s na 50Mbit/s a menej), vzhľadom na jedného pripojeného hosta.

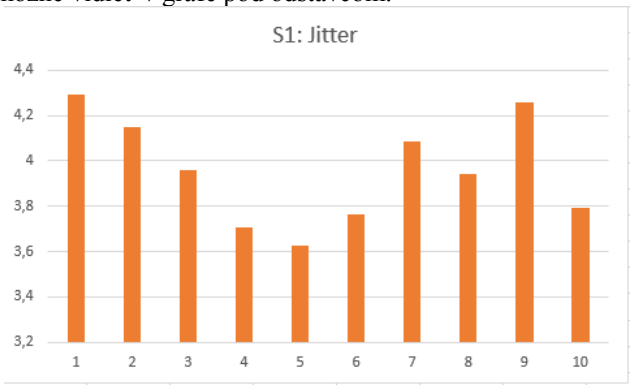
Potrebovali sme totižto zabezpečiť, aby nedošlo k zahlteniu linky inde, ako medzi prepínačmi. Nakoľko všetky linky mali prenosovú rýchlosť 100Mbit/s, jediná možnosť bola znížiť rýchlosť liniek medzi prepínačmi. Ďalším dôvodom bolo aj použitie virtuálneho hosta na fyzickom, čím sa zabezpečila komunikácia dvoch rôznych hostov cez jednu linku.

Z tohto dôvodu nie je možné porovnávať meranie na fyzickej topológií s tým v Mininete, ba ani s tým reálnym, ktoré namerali autori pôvodného článku. Na druhú stranu ako ukážkové zapojenie a otestovanie reálneho prostredia, to však pre nás pridanú hodnotu má.

(1) Základný test

Ako sme očakávali, pri základnom teste, kedy na pozadí nebeží žiadny iný tok, nám vyšli najlepšie možné hodnoty percentuálnej stratovosti paketov a to 0%. To znamená, že každý jeden paket nám prešiel úspešne zo zdroja do cieľa.

Čo sa týka hodnôt jitter-u, tie v danom prípade vychádzali v rozpätí od 3,625 ms až po 4,295 ms, čo nám vytvorilo priemernú hodnotu 3,9576 ms. V porovnaní z reálnymi hodnotami, ktoré namerali autori 0,0097 ms, sú tieto hodnoty úplne iné, avšak môžu za to aj rozdielne zariadenia (autor - Cisco Catalyst 3650; my - Soekris net6501) a taktiež aj náležitosti, ktoré som spomínal vyššie. Nami namerané hodnoty je možné vidieť v grafe pod odstavcom.



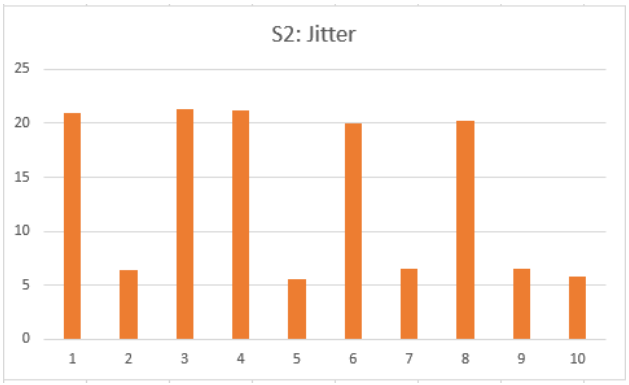
Obr. 15 - Jitter v reálnom prostredí v základom teste

(2) Výkonnostný test bez DTD

Pri výkonnostnom teste bez DTD nám vyšla veľmi podobná stratovosť paketov ako autorovi a to 55,3%. Autorovi táto priemerná percentuálna stratovosť vyšla 50%. Je vidno, že rozdiel je veľmi malý a činí len 5,3%. V porovnaní rovnakého testu, avšak v prostredí mininetu nie je tento rozdiel taktiež priepastný, ba naopak celkom podobný: 55,3% ku 40,2728% (Rozdiel cca 15%). Namerané hodnoty možno vidieť v Obr. 16.

Obr. 16 - Stratovosť paketov v reálnom prostredí vo výkonnostnom teste

V teste merania jitter-u sme tiež takmer dosiahli veľmi podobné hodnoty ako autor. Priemerná hodnota jitter-u v reálnom prostredí, ktorú autor nameral činí 7,829 ms a naša je 13,4523 ms. Ostatné hodnoty je možné vidieť pod odstavcom.

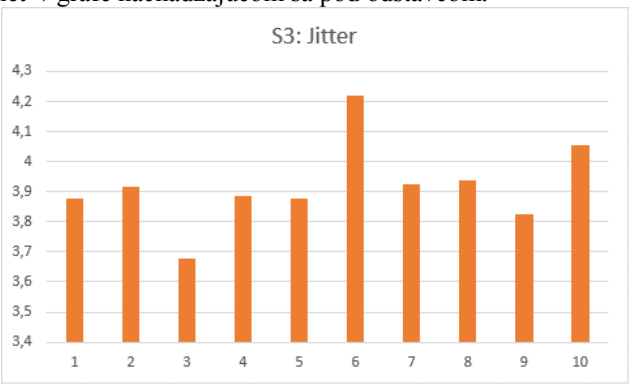


Obr. 17 - Jitter v reálnom prostredí vo výkonnostnom teste

(3) Výkonnostný test s použitím DTD

Čo sa týka stratovosti paketov vo výkonnostnom teste s použitím DTD na reálnych zariadeniach, nám vyšla hodnota stratovosti paketov 0,0%. Tento reálny prípad nám vyšiel rovnako ako autorovi práce a okrem iného dokonca lepšie ako v emulátore Mininet (tam dosahoval zanedbateľné hodny - 0,00005%).

Meranie jitter-u aj keď nám vrátilo priemernú hodnotu 3,9196 ms, čo sa celkom s autorom nezhoduje (on nameral 0,0097 ms) nám toho veľa vypovedá. Rozdiely medzi našimi testami (scenár 1 a 3) nám vrátili takmer rovnakú hodnotu s rozdielom len 0,038 ms. Podobný rozdiel sa vyskytol aj medzi autorovými meraniami. Všetky čiastkové merania je možné vidieť v grafe nachádzajúcom sa pod odstavcom.



Obr. 18 - Jitter v reálnom prostredí vo výkonnostnom teste s použitím DTD

IV. Zhodnotenie a záver

Hlavnou časťou našej práce bolo analyzovať postupy a výsledky dosiahnuté autormi pôvodného článku. Na základe tejto analýzy sme vypracovali návrh nášho riešenia daného problému na základe už existujúceho riešenia, ktoré sme ale v prípade hardvérovej časti upravili podľa toho, koľko a akých zariadení sme mali k dispozícii. Vytvorili sme si vlastné skripty, či už na vytvorenie samotnej mininet topológie alebo na vykonávanie DTD algoritmu.

Následne sme vykonali rovnakú sériu testov a výsledky porovnali, najmä s výsledkami z referenčného článku, nakoľko porovnanie hardvérovej a emulovanej topológie by nebolo vhodné. Došlo totižto k podstatnej úprave reálnej topológie, a teda cieľom hardvérovej časti bolo vyskúšať si, ako funguje SDN na reálnom zariadení a či vôbec to bude fungovať.

V práci sa nám potvrdilo to, čo sme aj predpokladali a čo predpokladali aj autori článku. DTD algoritmus pri nežiadúcej premávke (zahltená primárna cesta) dynamicky vytvára záložnú cestu, čo znižuje stratovosť paketov a taktiež aj jitter. Táto hypotéza sa nám aj prakticky potvrdila pri našich zreplikovaných testovaniach, ktoré sme vzhľadom na HW ktorým sme disponovali museli sčasti upraviť.

Aj napriek úpravám, ktoré sme zaviedli sa výsledky do výraznej miery zhodovali s tými, ktoré namerali autori článku. Okrem iného sa s časti zhodovali aj merania v rámci rovnakých scenárov v prostredí reálnom a Mininet. To že merania častokrát dosahovali lepšie hodnoty v emulátore Mininet, je spôsobené aj tým, že Mininet patrí medzi emulátori typu "All in one" (všetko na jednom mieste - žiadne prepojovacie káble a iné HW oneskorenia).

Vďaka projektu sme sa naučili pracovať s emulátorom Mininet, HW - Soekris net6501, s SDN kontrolérom RYU a v neposlednom rade si rozšírili naše programátorské znalosti, tímovú prácu s Git-om ale aj veľa ďalšieho. Osobne považujeme projekt ako veľmi prínosný a radi by sme v ňom pokračovali. Predsa len bolo by dobré pomocou reálneho prostredia vytvoriť topológiu totožnú s emulovanou, skúsiť nastaviť odozvy emulovaných liniek na rovnakú hodnotu s tými reálnymi a tak preukázať, že mininet je veľmi mocný nástroj, pomocou ktorého je možno emulovať a testovať SDN siete s dosiahnutím dôveryhodných výsledkov.

References

[1] BARRETT, Robert, et al. Dynamic Traffic Diversion in SDN: testbed vs Mininet. In: Computing, Networking and Communications (ICNC), 2017 International Conference on. IEEE, 2017. p. 167-171 (http://ieeexplore.ieee.org/document/7876121/references).

[2] HALEPLIDIS, Evangelos, et al. Software-defined networking (SDN): Layers and architecture terminology. 2015 (https://www.rfc-editor.org/rfc/pdfrfc/rfc7426.txt.pdf).

[3] Schroder Carla, Measure Network Performance with iperf [online]. Január 2007. Dostupné na internete: <http://www.enterprisenetworkingplanet.com/netos/article.php/3657236/Measure-Network-Performance-with-iperf.htm>

[4] Iperf - The TCP/UDP Bandwidth Measurement Tool, Online: (http://sourceforge.net/projects/iperf2/)