# Analiza karakteristika i primjene softverskih definiranih mreža

**Pušeljić, Anthony**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET PROMETNIH ZNANOSTI**

Anthony Pušeljić

# ANALIZA KARAKTERISTIKA I PRIMJENE SOFTVERSKI DEFINIRANIH MREŽA

DIPLOMSKI RAD

Zagreb, rujan 2015.

Sveučilište u Zagrebu

Fakultet prometnih znanosti

DIPLOMSKI RAD

# ANALIZA KARAKTERISTIKA I PRIMJENE SOFTVERSKI DEFINIRANIH MREŽA

# ANALYSIS OF CHARACTERISTICS AND APPLICATION OF SOFTWARE DEFINED NETWORKS

Mentor: dr. sc. Ivan Grgurević

Student: Anthony Pušeljić, 0036429747

Datum obrane: 24. rujna 2015.

Zagreb, rujan 2015.

**ANALIZA KARAKTERISTIKA I PRIMJENE SOFTVERSKI DEFINIRANIH MREŽA**


## SAŽETAK

Računalne mreže su se razvile zbog novih trendova i zahtjeva korisnika za što bržom uspostavom i isporukom usluga. Nefleksibilnost arhitekture računalnih mreža predstavlja izazov razvijateljima jer se njihovi eksperimenti ne mogu ocijeniti u stvarnim mrežama. Softverski definirana mreža (SDN) i *OpenFlow* arhitektura omogućuju način implementacije programabilnih mrežnih arhitektura koje se mogu implementirati postepeno u već postojeću mrežu. Kontrola mreže je na vanjskom uređaju u obliku *software*-a odvojena od prosljeđivanja paketa i ima mogućnost direktnog programiranja. SDN omogućuje dinamičku prilagodbu mrežnog okruženja trenutnim aplikativnim zahtjevima ili potrebama korisnika te znatno pojednostavljuje upravljanje i povećava skalabilnost mreže. U ovom radu su analizirane funkcionalnosti i karakteristike softverski definiranih mreža i na temelju usporedbe s konvencionalnom mrežom utvrđene su bitne razlike, prednosti i nedostaci te koliko SDN utječe na samo poslovanje jedne organizacije (tvrtke).



**KLJUČNE RIJEČI:** Softverski definirana mreža (SDN); *OpenFlow*; kontroler, agilnost; fleksibilnost; *Application programming interface* (API)



## SUMMARY

The new trends and users' demands for fastest setting up and delivering of new services are the main reasons for computer networks being developed. The inflexibility in architecture of computer networks represents a challenge for developers due to the fact that their experiments can not be verified in real networks. Software Defined Networks (SDN) and the OpenFlow architecture allow implementation of programmable networks' architectures, which can be gradually deployed in already existing networks. The network control is on an external device as a kind of software program (Controller). It is separated from the packet forwarding (Switch), having the possibility of being directly programmed. SDN enables dynamic adaptation of networks' environments to the current applications' requests or users' demands, making the network control easier and at the same time increasing its scalability. In this thesis, the functionality and characteristics of software defined networks have been analysed and due to the comparison with conventional networks, the main differences, advantages and drawbacks have been determined as well as the answer to the question: How much software defined networks can influence the core business of an organisation?



**KEYWORDS:** Software Defined Network; OpenFlow; Controller; agility; flexibility; Application programming interface (API)

# SADRŽAJ

# 1. UVOD

U današnje vrijeme svjedoci smo vrlo visokog stupnja primjene virtualizacijskih tehnologija i tehnika uz rastuće zahtjeve korisnika za što bržom uspostavom i isporukom usluga te smještanja usluga u sklopu koncepta Računarstva u oblaku (engl. *Cloud Computing*). Uz navedeno, korisnici zahtijevaju fleksibilno i automatizirano mrežno okruženje koje je prilagodljivo trenutnim aplikativnim zahtjevima. Na ovakve nove izazove gotovo je nemoguće odgovoriti primjenom klasičnog upravljanja mrežnom infrastrukturom. Računarstvo u oblaku korisnicima omogućuje pohranu podataka i instalaciju programske podrške na poslužitelje koji su povezani putem Interneta. Uz pomoć web preglednika i posebnih klijenata, ove su usluge fleksibilne, a korisnici plaćaju samo ono što koriste.

Softverski definirana mreža (engl. *Software-defined network*/*networking*, kratica SDN) je mrežna arhitektura, gdje je kontrola mreže odvojena od proslijeđivanja paketa i ima mogućnost direktnog programiranja. Takva migracija kontrole, nekada čvrsto vezane za pojedinačni mrežni uređaj, u vanjske računalne uređaje omogućava osnovnoj infrastrukturi odvojenost od aplikacija i mrežnih usluga, koje sada mogu tretirati mrežu kao logički ili virtualni entitet. SDN omogućuje dinamičku prilagodbu mrežnog okruženja trenutnim aplikativnim zahtjevima ili potrebama korisnika te znatno pojednostavljuje upravljanje i povećava skalabilnost mreže, što se posebno očituje vrlo jednostavnom implementacijom dodatnih mrežnih servisa (usluga) i komponenti. Dodatna prednost SDN-a je mogućnost korištenja mrežnih komponenti različitih proizvođača, praktički bez potrebe poznavanja rada na samim uređajima, jer se kompletnim mrežnim okruženjem upravlja iz jedne točke, odnosno putem SDN kontrolera (engl. SDN *Controller*). SDN mrežna arhitektura sastoji se od SDN kontrolera, *OpenFlow* mrežnih uređaja i *OpenFlow* komunikacijskog kanala koji ih povezuje.

Diplomski rad se sastoji od osam povezanih poglavlja:

1. Uvod,
2. Osnovne karakteristike softverski definiranih mreža (SDN),
3. Pregled arhitekture softverski definiranih mreža,

4. Uloga OpenFlow protokola kao temeljnog elementa softverski definiranih mreža,
5. Sigurnost softverski definiranih mreža,
6. Planiranje softverski definiranih mreža,
7. Primjena i simulacijski prikaz rada softverski definirane mreže,
8. Zaključak.

Svrha istraživanja je prikazati funkcionalnosti i karakteristike softverski definiranih mreža, kako se primjenjuju i na koji se način gradi nova softverski definirana mreža te kakve su mogućnosti implementacije u već postojeću konvencionalnu mrežu.

Cilj istraživanja je provesti analizu karakteristika i primjene softverski definiranih mreža. Analiza će se temeljiti na usporedbi konvencionalnih mreža i softverski definiranih mreža uz prikaz bitnih razlika. Utvrditi će se da li i u kojoj mjeri softverski definirana mreža utječe na samo poslovanje te koje su prednosti i nedostaci softverski definirane mreže u odnosu na konvencionalnu mrežu, uzevši u obzir različite parametre i dionike.

Istraživanje uključuje simulaciju različitih mrežnih topologija pomoću programske podrške odnosno simulatora na operacijskom sustavu *Linux*. Za potrebe simulacije prikazati će se razlike mrežne arhitekture jedne konvencionalne mreže i softverski definirane mreže te će se putem simulatora prethodno provesti postupak izgradnje softverski definirane mreže.

# 2. OSNOVNE KARAKTERISTIKE SOFTVERSKI DEFINIRANIH MREŽA (SDN)

*Software Defined Networking* (SDN) je novi trend tehnologija koji je nastao potrebom da se odvaja i redefinira mrežna konstrukcija te je cilj bolje upravljanje mrežama većih razmjera i složenosti. Uz eksplozivan rast podatkovnog prometa u zadnjih nekoliko godina, konvencionalne podatkovne mreže bile su izložene zagušenjima. *Software Defined Networking* je pokušaj rješavanja zagušenja konvencionalnih implementacija *switching*/*routing-based* umrežavanja te koristi slijedeća tri načela:

1. *Control and forwarding planes*

   **Konvencionalna mreža**: Kontrolna cjelina i cjelina prosljeđivanja se zajedno nalaze u mrežnim elementima.

   **SDN**: Kontrolna cjelina je odvojena od cjeline proslijeđivanja i premještena do SDN kontrolera.
2. *Control intelligence*

   **Konvencionalna mreža**: Kontrolna inteligencija raspoređena je u svakom mrežnom elementu.

   **SDN**: Kontrolna inteligencija centralizirana je na SDN kontroleru.
3. *Network programmability by applications*

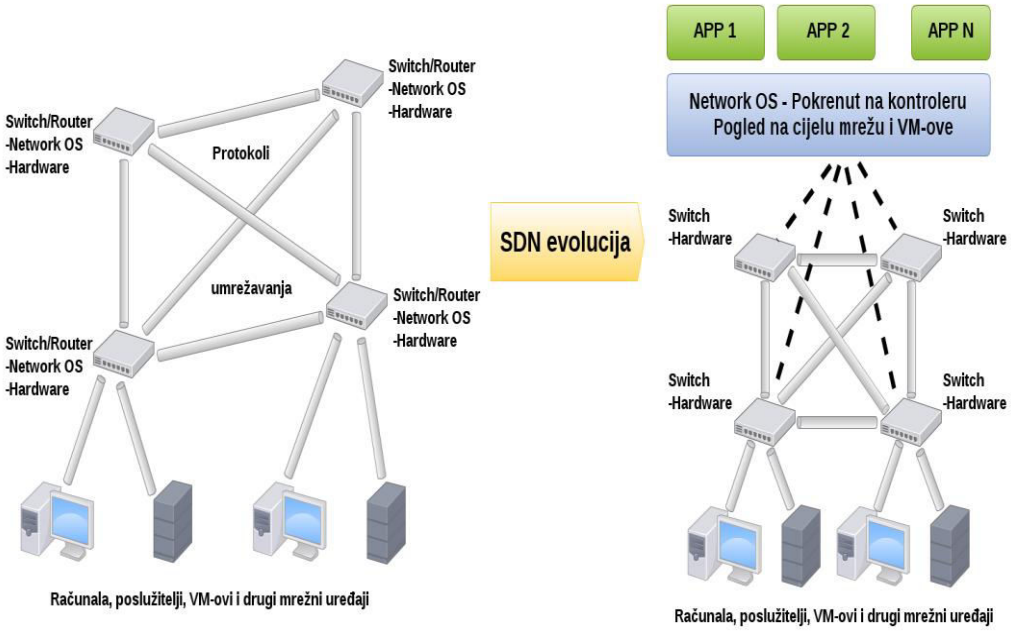
   **Konvencionalna mreža**: Mreža ne može biti programirana od aplikacija. Svaki mrežni element mora biti posebno konfiguriran.

   **SDN**: Mreža može biti programirana od aplikacija. Sučelja aplikacija mogu biti izloženi kontroleru za manipulaciju mreže.

Do sada je bilo prisutno nekoliko implementacija različitih proizvođača. Neki proizvođači usredotočeni su na *OpenFlow-based* kontrolere, a neki rade na razvoju mreže preko gledišta virtualnog stroja (apstrahiranje mrežnih *switch*-eva), kao što su *VxLAN-based*[1] rješenja [1]. U osnovi, SDN je nešto što zadovoljava gore navedena tri uvjeta. Na slici 1. prikazana je definicija konvencionalne mreže i SDN-a iste mreže te njena lijeva strana predstavlja trenutne mrežne implementacije koje obično imaju

[1] VxLAN: Virtual Extensible LAN (VxLAN) je tehnologija virtualizacije mreže koja pokušava poboljšati probleme skalabilnosti povezanih s velikim Cloud Computing implementacijama

*switch*-eve i *router*-e koji su međusobno povezani jedni s drugima te uređaje (poslužitelji/virtualni strojevi) na rubu mreže. Sadašnje mrežne implementacije oslanjaju se na korištenje distribuiranih protokola za izgradnju kontrolnog puta. Nakon što je kontrolni put napravljen, podatkovna cjelina instalirana je na *hardware*-u i običan promet prolazi na programiranom putu. U sadašnjim implementacijama ti kontrolni protokoli raspoređeni su preko mreže kako bi napravili kontrolne i podatkovne staze koje su uglavnom l*ayer-2* i *layer-3* protokoli, kao što su BGP, OSPF, STP i drugi [2].




Slika 1. Definicija konvencionalne mreže i definicija SDN-a iste mreže

Izvor: Izradio autor

Desna strana slike 1. prikazuje definiciju SDN-a iste mreže, pri čemu će se *Network Operating System* (OS) izvoditi na jednom entitetu (npr. kontroleru), a sve odluke protoka biti će donesene putem tog entiteta te će nakon donošenja odluke, kontroler programirati podatkovnu cjelinu *switch*-a. Isprekidana crta na desnoj strani slike 1. prikazuje komunikaciju između kontrolera i mrežnih uređaja. Na tim linkovima, kontroler može imati globalni pogled na cjelokupnu mrežu. Neprekidne crte su redoviti podatkovni linkovi na kojima će podaci teći nakon što je *switching*/*routing hardware* instaliran.

Analizirajući problematiku SDN-a, tvrtka Cisco došla je do zaključka da SDN uvelike pomaže pri pojednostavljenju poslovanja automatizacijom i centralizacijom mrežnog poslovnog upravljanja [3].

Cilj SDN-a je osigurati da su sve logičke odluke kontrolne razine donesene na jednom centralnom mjestu, u odnosu na konvencionalno umrežavanje, gdje su odluke kontrolne razine donesene lokalno i inteligencija je distribuirana u svakom *switch*-u. Imajući ovaj centralni pristup, smanjit će se potreba za N-broj inteligentnih čvorova u topologiji s N-čvorova. Osnovna uporaba bilo kojeg mrežnog *software*-a je programirati put, tako da promet može teći na tom određenom putu. Sada, kada je ovisnost *software*-a na *hardware*-u smanjena, nema potrebe da se inteligentni *software* pokreće na svim čvorovima. SDN se oslanja na koncept logičkog pokretanja *software*-a na centraliziranom mjestu i programiranjem *switch*-eva pomoću *southbound Application program interfaces-a*[2] (kratica API).

*SDN Controller*

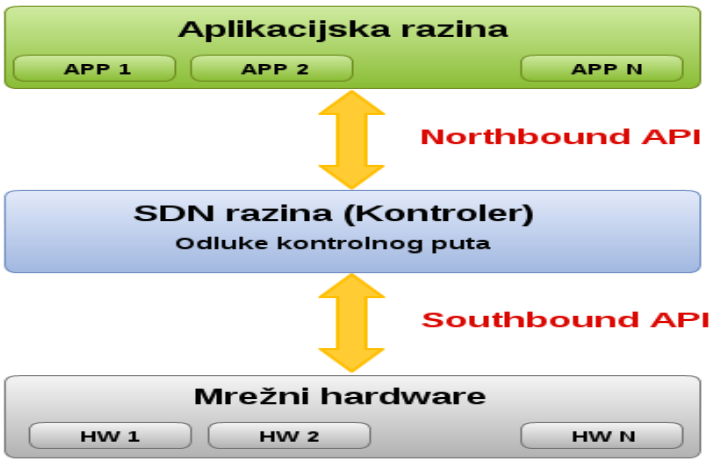
Središnji kontroler (SDN kontroler) je entitet *software*-a koji treba imati globalni pogled na cijelu mrežu (uključujući VM-e i prometne tokove). Kao što je objašnjeno na slici 1., mrežni operacijski sustav, koji je pokrenut logički za izbor puta, treba biti pokrenut na ovom središnjem SDN kontroleru. Zbog toga što će kontroler imati pogled na cijelu mrežu, može odlučiti o optimiziranom protoku i programirati ulaze u *hardware*-u. Trenutno većina SDN kontrolera ima grafičko korisničko sučelje (engl. *Graphical User Interface*, kratica GUI), koje može administratoru dati slikovni pogled na cijelu mrežu.

Kontroleri se također gledaju kao platforme koje se mogu koristiti za pokretanje aplikacija za kontrolu mreže. Ove aplikacije ne moraju znati složenu fizičku infrastrukturu ispod kontrolera. Tipični primjeri tih aplikacija su analize protoka ili programiranje mreže koje se okida određenim događajem. Može biti više od jednog kontrolera u mreži te je stoga podržana visoka dostupnost kontrolne cjeline. Kada postoji više od jednog kontrolera u mreži, događa se redovita sinkronizacija između njih. Više o SDN kontroleru biti će objašnjeno u sljedećem poglavlju.

[2] API: Application programming interface ili sučelje za programiranje aplikacija je skup određenih pravila i specifikacija koje programeri slijede tako da se mogu služiti uslugama ili resursima operacijskog sustava ili nekog drugog složenog programa kao standardne biblioteke rutina (funkcija, procedura, metoda), struktura podataka, objekata i protokola.

*Logical Layers of SDN*

Slika 2. prikazuje logičke slojeve SDN-a. Na najnižem dijelu se nalaze mrežni elementi kao što su *switch*-evi, računala, poslužitelji i ostali mrežni uređaji. Treba naglasiti da se *switch*-evi nalaze na vrhu najnižeg sloja. Srednji sloj je sloj kontrolera koji komunicira s *switch*-evima. Najviši sloj je aplikacijski sloj u kojem korisnik može definirati aplikacije koje će okidati definiciju protoka u mreži.




Slika 2. Logički slojevi SDN-a

Izvor: Izradio autor

*Southbound API*

U arhitekturi softverski definirane mreže, *southbound* API-ji se koriste za komunikaciju SDN kontrolera sa *switch*-evima i *router*-ima mreže. *Southbound* API-ji olakšavaju učinkovitu kontrolu nad mrežom i omogućuju da SDN kontroler dinamički radi promjene prema zahtjevima i potrebama u stvarnom vremenu. *OpenFlow*, koji je razvijen od strane *Open Networking Foundation* (ONF), je prvo i vjerojatno najpoznatije *southbound* sučelje. To je industrijski standard koji definira način na koji SDN kontroler treba djelovati s cjelinom prosljeđivanja za prilagođavanje mreže da se bolje prilagodi promjenjivim poslovnim potrebama. S *OpenFlow*-om, zapisi se mogu dodati i ukloniti u internoj tablici protoka *switch*-eva i potencijalno *router*-a, da bi mreža bila više osjetljiva na zahtjeve u stvarnom vremenu.

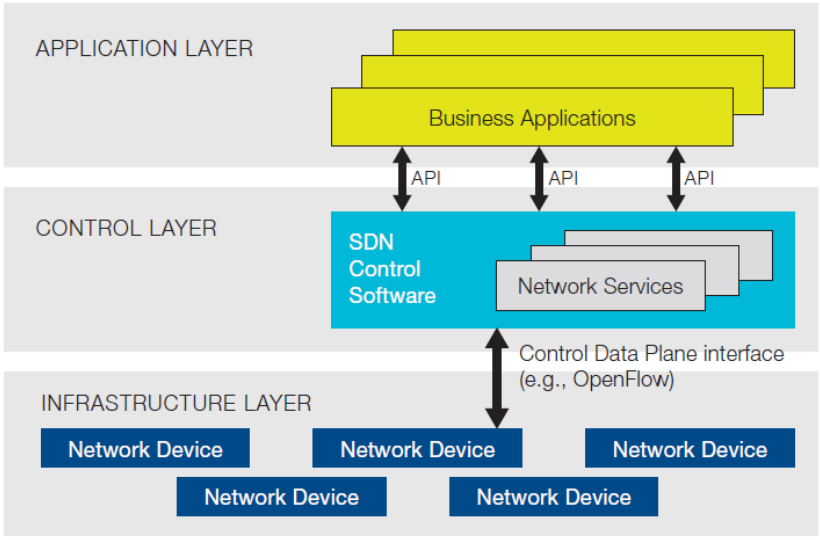
*Northbound API*

U SDN mreži, *northbound* API-ji se koriste za komunikaciju SDN kontrolera sa uslugama i aplikacijama koje su pokrenute u mreži. *Northbound* API-ji se mogu koristiti da bi se olakšale inovacije i omogućila učinkovita orkestracija i automatizacija mreže i da bi se ona uskladila s potrebama različitih aplikacija preko njene programabilnosti. *Northbound* API-ji su nedvojbeno najkritičniji API-ji u SDN okruženju, jer vrijednost SDN-a je vezana za inovativne aplikacije koje potencijalno može podržati i omogućiti te moraju podržavati široku paletu aplikacija [4].

# 3. PREGLED ARHITEKTURE SOFTVERSKI DEFINIRANIH MREŽA

Kao što je rečeno, *Software Defined Networking* je mrežna arhitektura gdje je kontrola mreže odvojena od proslijeđivanja te se može izravno programirati. Ova migracija kontrole, nekad čvrsto vezana u pojedinačnim mrežnim uređajima, u dostupnim računalnim uređajima omogućuje da se infrastruktura nižeg sloga izdvoji za aplikacije i mrežne usluge.

Slika 3. prikazuje logički pogled na SDN arhitekturu. Mrežna inteligencija je centralizirana u softverski baziranim SDN kontrolerima, što omogućuje globalni pogled na mrežu.




Slika 3. Logički pogled na SDN arhitekturu [5]

Kao rezultat toga, mreža pristupa aplikacijama kao jedan logički *switch* te se time dobije kontrola nad cijelom mrežom s jedne logičke točke, što pojednostavljuje dizajn mreže i same operacije unutar mreže. SDN također pojednostavljuje rad samih mrežnih uređaja, jer više ne trebaju razumjeti i obraditi tisuće protokolarnih standarda i samo trebaju prihvatiti upute od SDN kontrolera [6].

Možda je najvažnije da mrežni operatori i administratori mogu programski konfigurirati ovu pojednostavljenu apstrakciju mreže i ne moraju ručno podesiti desetaka tisuća linija koda za konfiguraciju koji su razasuti među tisućama uređaja. Osim toga, utjecajem centralizirane inteligencije SDN kontrolera, moguće je promijeniti ponašanje mreže u realnom vremenu i implementirati nove aplikacije i

usluge u nekoliko sati ili dana, a ne u nekoliko tjedana ili mjeseci. Centraliziranjem mrežnog stanja u kontrolnom sloju, SDN pruža mrežnim upraviteljima fleksibilnost u konfiguriranju, upravljanju i optimiziranju mrežnih resursa putem dinamičnih i automatiziranih SDN programa. Osim apstrahiranja mreže, SDN arhitekture podržavaju set API-ja koji omogućuju implementaciju zajedničkih mrežnih usluga, uključujući usmjeravanje, *multicast*, sigurnost, pristup kontroli, upravljanje propusnošću, prometno inženjerstvo, kvaliteta usluge, optimizacija procesora i skladištenja te potrošnju energije [2].

U sklopu analize SDN-a provedene od strane *Open Network Foundation* [5] navodi se da je rezultat odvojenih kontrolnih i podatkovnih cjelina veća programabilnost, automatizacija te bolja kontrola mreže, što omogućuje vrlo skalabilne i fleksibilne mreže kako bi se primjerice tvrtke lako prilagodile promjenjivim poslovnim potrebama.

Ključni principi SDN arhitekture su:

- Konvencionalne aplikacije (koje nisu SDN aplikacije), samo implicitno i neizravno opisuju svoje mrežne zahtjeve, što obično uključuje nekoliko koraka obrade.
- Konvencionalne mreže (npr. trenutni Internet i njegove usluge poput pregledavanja web stranica, medijski *streaming*) ne nude (dinamički) način izražavanja cijelog niza zahtjeva korisnika, primjerice propusnost, kašnjenje, varijacija kašnjenja ili dostupnost. Zaglavlja paketa mogu kodirati prioritetne zahtjeve, međutim mrežni *provider*-i obično ne vjeruju prometnim oznakama korisnika. Stoga neke mreže pokušavaju zaključiti korisničke zahtjeve prema vlastitim (npr. kroz analizu prometa), koji mogu uzrokovati dodatne troškove, a to ponekad dovodi do pogrešne klasifikacije. SDN nudi mogućnost za korisnika da u potpunosti odredi svoje potrebe u kontekstu pouzdanog odnosa.
- Konvencionalne mreže ne izlažu informacije i mrežno stanje aplikacijama koje ih koriste. Koristeći SDN pristup, SDN aplikacije mogu pratiti mrežno stanje i prilagoditi se u skladu s tim stanjem [4].

## 3.1. SDN Controller

Kontrolna cjelina je logički centralizirana i odvojena od podatkovne cjeline. SDN kontroler rezimira mrežno stanje za aplikacije i prevodi zahtjeve aplikacija za pravila niske razine:

- To ne znači da je kontroler fizički centraliziran. Za performanse, skalabilnost i/ili razloge pouzdanosti, logički centralizirani SDN kontroler može se distribuirati, tako da nekoliko fizičkih kontrolera mogu međusobno surađivati za kontrolu mreže i poslužiti aplikacije.
- Kontrolne odluke se donose na *up-to-date* globalnom pogledu mrežnog stanja. Sa SDN-om, kontrolna cjelina djeluje kao jedan, logički centralizirani mrežni operativni sustav u smislu raspoređivanja i rješavanja sukoba resursa, kao i udaljeno apstrahiranje pojedinosti uređaja niske razine, npr. električni prijenos protiv optičkog prijenosa.

Temeljne značajke svakog kontrolera su:

- Otkrivanje uređaja krajnjih korisnika kao što su prijenosna računala, stolna računala, printeri, mobilni uređaji itd.
- Otkrivanje mrežnih uređaja koje čine infrastrukturu mreže, kao što su *switch*-evi, *router*-i i bežične pristupne točke.
- Upravljanje topologijom mrežnih uređaja održavanjem informacija o detaljima povezanosti između mrežnih uređaja i krajnjih uređaja na koje su direktno vezani.
- Upravljanje protokom održavanjem baze podataka protoka kojima upravlja kontroler i obavljanje svih potrebnih koordinacija s uređajima kako bi se osigurala sinkronizacija *flow entry*-a uređaja s tom bazom podataka [7].

Sve su implementirane skupinom internih modula u kontroleru. Ti moduli trebaju održavati lokalne baze podataka koje sadrže trenutnu topologiju i statistike. Kontroler prati topologiju učenjem stanja *switch*-eva i uređaja krajnjih korisnika te

prati povezanost između njih. Održava *flow cache*[3] koji zrcali tablice protoka na raznim *switch*-evima koje kontrolira i lokalno održava *per-flow* statistike prikupljenih iz *switch*-eva.

SDN kontroler ima potpunu kontrolu nad podatkovnim putevima, ovisno o granici njihovih mogućnosti te se stoga ne mora natjecati/boriti s drugim elementima kontrolne cjeline, što olakšava planiranje i upravljanje resursima. To omogućuje mrežama da se izvode sa složenim i preciznim pravilima, s većom iskoristivošću resursa i jamstvom kvalitete usluge. To se događa kroz dobro razumljivi zajednički informacijski model (npr. kao onaj definiran od strane *OpenFlow*-a).

U teoriji, SDN kontroler pruža usluge koje mogu ostvariti distribuiranu kontrolnu cjelinu, u stvarnosti, bilo koji primjer kontrolera pružati će jedan dio ili podskup funkcionalnosti. Opći opis jednog SDN kontrolera je softverski sustav ili zbirka sustava koji zajedno pružaju:

- Upravljanje mrežom, a u nekim slučajevima, upravljanje i distribucija može uključivati bazu podataka. Te baze podataka služe kao spremište za informacije koje proizlaze iz kontrolnih mrežnih elemenata i povezanog *software*-a, kao i informacije kontrolnih SDN aplikacija, uključujući mrežno stanje, neke informacije o konfiguraciji, saznanja o topologiji i informacije kontrolne sesije.
- Podatkovni model na visokoj razini koji bilježi odnose između upravljanim resursima i ostalih usluga koje pruža kontroler. U mnogim slučajevima, ti podatkovni modeli izgrađeni su pomoću *Yang* jezika koji služi za modeliranje.
- Moderni API je uvjet koji izlaže usluge kontrolera aplikaciji te olakšava većinu *controller-to-application* interakcija. Ovo sučelje je idealno doneseno iz podatkovnog modela koji opisuje usluge i značajke kontrolera. Neki sustavi idu korak dalje i osiguravaju robusne razvojne okoline koje omogućuju proširenje temeljnih sposobnosti i naknadno objavljivanje API-ja za nove module, uključujući i one koji podržavaju dinamično proširenje sposobnosti kontrolera.
- Sigurna TCP kontrolna sesija između kontrolera i pripadajućih agenata u mrežnim elementima.

[3] Cache: Priručna memorija, predmemorija ili *cache* je mala memorija koja služi za pohranu podataka koji se često koriste. U nju se, za razliku od buffer memorije može pisati na koje mjesto se želi i čitati s kojeg mjesta se želi.

- Standardni bazirani protokol za *application-drive* mrežno stanje na mrežnim elementima.
- Uređaj, topologija i mehanizam za otkrivanje usluga; sustav za računanje puta i potencijalno druge informacijske usluge koje su orijentirane mreži i resursu.

Trenutni izbor kontrolera uključuje komercijalne proizvode kao i niz *open source* kontrolera, poput *VMware* (*vCloud*/*vSphere*), *Nicira* (*NVP*), *NEC* (*Trema*), *Big Switch Networks* (*Floodlight*/BNC) i *Juniper*/*Contrail*.

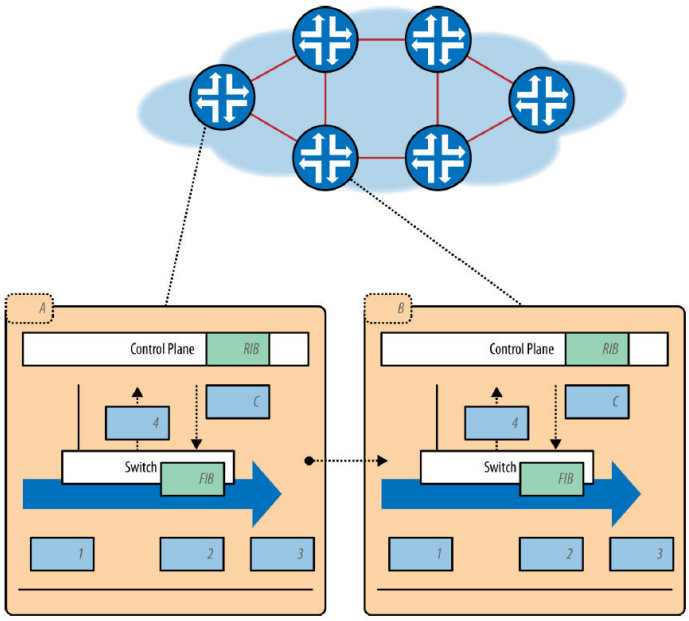
## 3.2. Control and Data Plane

Odvajanje kontrolne i podatkovne cjeline je jedan od temeljnih načela SDN-a. Ovo odvajanje pruža mrežnom operatoru određene prednosti u odnosu na centraliziranu ili polu-centraliziranu programsku kontrolu. Također ima i potencijalnu ekonomsku prednost na temelju sposobnosti da se konsolidira na jednom mjestu ili nekoliko mjesta, što je često značajan složen dio *software*-a za konfiguriranje i kontrolu *hardware*-a. U nastavku će se objasniti pojedine cjeline.

Na vrlo visokoj razini, kontrolna cjelina uspostavlja lokalni skup podataka koji se koristi za izradu ulaza tablice prosljeđivanja, koje koristi podatkovna cjelina za prosljeđivanje prometa između ulaznih i izlaznih portova na uređaju. Skup podataka koji se koriste za pohranu mrežne topologije naziva se *Routing Information Base* (RIB). RIB se često čuva putem razmjene informacija između ostalih instanci kontrolne cjeline unutar mreže. Ulazi tablice prosljeđivanja se obično nazivaju *Forwarding Information Base* (FIB), a često se zrcale između kontrolne i podatkovne cjeline uređaja. Za izvođenje ovog zadatka, kontrolni entitet/program mora razviti pogled na topologiju mreže koji zadovoljava određena ograničenja. Ovaj pogled na mrežu može se programirati ručno, naučeno kroz promatranje ili izgraditi od dijelova informacija prikupljenih kroz diskurs s drugim instancama kontrolne cjeline, što može biti kroz korištenje jednog ili više protokola usmjeravanja, ručnog programiranja ili kombinacijom jednog i drugog. Mehanika kontrolne i podatkovne cjeline prikazano je na slici 4., koja predstavlja mrežu međusobno povezanih *switch*-eva [2].

Na vrhu slike, prikazana je mreža *switch*-eva, s proširenjem detalja kontrolne i podatkovne cjeline dvaju *switch*-a (navedeno kao A i B). Na slici, paketi su primljeni

od *switch*-a A na lijevoj strani kontrolne cjeline te su proslijeđeni prema *switch*-u B na desnoj strani slike. Unutar svake ekspanzije, kontrolna i podatkovna cjelina su odvojeni, međutim nalaze se u istom kućištu. Paketi su primljeni na ulaznim portovima *line card*-a gdje se podatkovna cjelina nalazi. Ako je npr. paket primljen koji dolazi od nepoznate MAC adrese, preusmjerava se (4) kontrolnoj cjelini uređaja, gdje se nauči, obrađuje, a zatim kasnije prosljeđuje dalje.




Slika 4. Kontrolna i podatkovna cjelina jednostavne mreže [2]

Taj isti tretman se daje kontrolnom prometu, kao što su *routing protocol messages*. Jednom kada je paket dostavljen kontrolnoj cjelini, podaci koji se nalaze u paketu se obrađuju i eventualno rezultiraju promjenom RIB-a, kao i prijenos dodatnih poruka svojim *peer*-ovima, kako bi ih upozorili zbog ažuriranja (tj. nova ruta je naučena). Kad RIB postane stabilan, FIB se ažurira u kontrolnoj i podatkovnoj cjelini. Nakon toga, prosljeđivanje će se ažurirati. Međutim, u ovom slučaju, pošto je primljeni paket bio jedan od nepoznate i nenaučene MAC adrese, kontrolna cjelina vraća paket (C) podatkovnoj cjelini (2), koja prosljeđuje paket (3). Ako je potrebno dodatno FIB programiranje, to se također odvija u (C) koraku. Isti algoritam za obradu paketa događa se u slijedećem *switch*-u desno.

*Layer* 2 kontrolna cjelina fokusirana je na *hardware* ili *physical layer* adrese poput IEEE MAC adresa. *Layer* 3 kontrolna cjelina je izgrađena kako bi se olakšale *network layer* adrese kao što su adrese IP protokola. *Layer* 2 i *layer* 3 problemi

skaliranja i njihovi rezultirajući dizajni kontrolne cjeline se eventualno spajaju ili križaju, jer *layer* 2 mreže ne skaliraju dobro zbog velikog broja krajnjih *host*-ova[4]. Jedan od glavnih problema, koji se bavi kretanjem krajnjih *host*-ova između mreža, rezultira masivnim gubitkom tablica prosljeđivanja i da ih se dovoljno brzo ažurira da ne ometaju protok prometa. U *layer* 2 mreži, prosljeđivanje se usredotočuje na dostupnost MAC adresa. Prema tome, *layer* 2 mreže prvenstveno se bave skladištenjem MAC adresa u svrhu prosljeđivanja. Kako MAC adrese *host*-ova mogu biti ogromne u velikoj mreži poduzeća, upravljanje tih adresa je otežano.

U *layer* 3 mreži, prosljeđivanje se usredotočuje na dostupnost mrežnih adresa. *Layer* 3 informacija dostupnosti prvenstveno se sama brine o dostupnosti odredišta IP prefiksa. To uključuje mrežne prefikse preko brojnih adresa za *unicast* i *multicast*. U svim suvremenim slučajevima, *layer* 3 umrežavanje koristi se za segmentiranje ili povezivanje *layer* 2 domena kako bi se prevladavali problemi *layer* 2 skaliranja. Specifično, *layer* 2 mostovi koji predstavljaju neke skupove IP podmreža, obično su međusobno povezani s *layer* 3 *router*-om. *Layer* 3 *router*-i međusobno su povezani da formiraju veće mreže - ili jako različite raspone adresa podmreže. Veće mreže se povezuju s drugim mrežama preko *gateway router*-a. Međutim, u svim tim slučajevima, *router* usmjerava promet između mreža na *layer* 3 te će proslijediti pakete na *layer* 2 kada zna da je paket stigao na konačno odredište *layer* 3 mreže, koji onda mora biti dostavljen do određenog *host*-a [7].

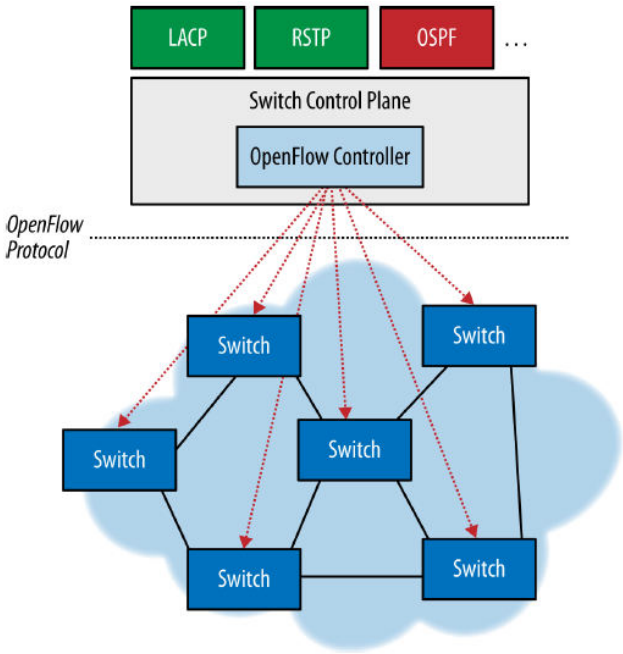
Podatkovna cjelina obrađuje dolazne datagrame kroz niz veza na razini operacija koje prikupljaju datagrame i obavljaju osnovne provjere. Dobro oblikovan (tj. točan) datagram se obrađuje u podatkovnoj cjelini obavljanjem *lookup*-ima u FIB tablici koji su ranije programirani od kontrolne cjeline. To se ponekad naziva i brzim putem za obradu paketa, jer ne treba daljnje ispitivanje osim identifikacije odredišta paketa pomoću programiranog FIB-a. Jedina iznimka kod ove obrade je kada se paketi ne podudaraju s tim pravilima, kao kada je otkriveno nepoznato odredište te se ti paketi šalju prema *route processor*-u gdje ih kontrolna cjelina može dodatno obraditi pomoću RIB-a [2].

[4] Host: Host je bilo koji uređaj povezan u računalnu mrežu (najčešće Internet) a koji može korištenjem standardnih protokola ostvariti komunikaciju s drugim sličnim uređajima (host-ovima). Host je u tom kontekstu najčešće konkretno osobno računalo, ali može biti i poslužitelj, usmjerivač, odnosno bilo koji uređaj koji ima mogućnost komunikacije. U kontekstu internet protokola (IP), host može biti bilo koji uređaj koji ima dodijeljenu IP adresu.

# 4. ULOGA OPENFLOW PROTOKOLA KAO TEMELJNOG ELEMENTA SOFTVERSKI DEFINIRANIH MREŽA

*OpenFlow* je izvorno zamišljen i implementiran kao dio istraživačke mreže na Sveučilištu *Stanford*. Njegov izvorni fokus je bio omogućavanje stvaranja pokusnih protokola na mreži kampusa koje se mogu koristiti za istraživanja i eksperimentiranja. Iz te početne ideje razvio se stav da bi *OpenFlow* mogao zamijeniti u potpunosti funkcionalnost *layer-2* i *layer-3* protokola u komercijalnim *switch*-evima i *router*-ima.

U 2011. godini, neprofitni konzorcij pod nazivom *Open Networking Foundation* (ONF) formiran je od skupina pružatelja usluga kako bi komercijalizirali, standardizirali i promicali korištenje *OpenFlow* u mrežama.




Slika 5. *OpenFlow* arhitektura [2]

Ključne komponente *OpenFlow* modela, kao što je prikazano na slici 5., postali su uglavnom dio zajedničke definicije SDN-a:

- Odvajanje kontrolne i podatkovne cjeline (u slučaju ONF-a, kontrolna cjelina se upravlja sa logički centraliziranog kontrolnog sustava),
- Korištenje standardiziranog protokola između kontrolera i agenta te
- Pružanje mrežnog programiranja iz centraliziranog pogleda.

*OpenFlow* je skup protokola i API-ja te su trenutno podijeljeni u dva dijela:

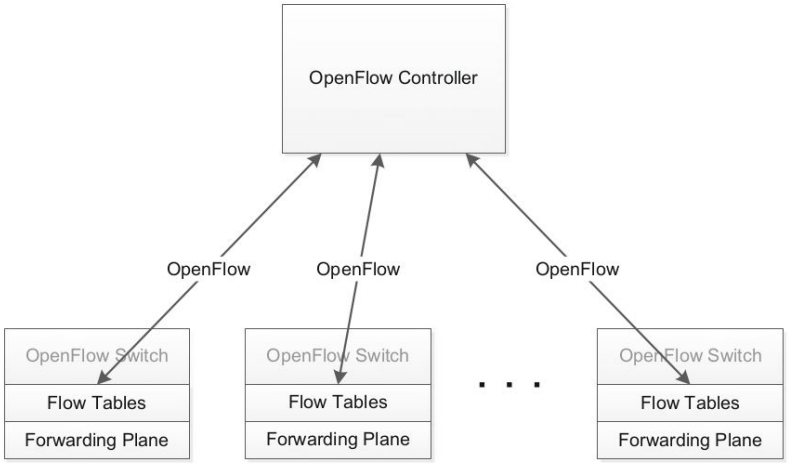
- *Wire protocol* za uspostavu kontrolne sesije, definiranje strukture poruke za razmjenu promjene toka (*flowmods*) i prikupljanje statistika i definiranje temeljnih struktura *switch*-a (portovi i tablice),
- Konfiguracijski i upravljački protokol za dodjeljivanje fizičkih portova *switch*-a određenom kontroleru, definiranje visoke dostupnosti (*active*/*standby*) i ponašanja kod neuspjelog povezivanja s kontrolerom.

## 4.1. Općenito o OpenFlow protokolu

*OpenFlow* protokol je zapravo *southbound* protokol i sličan bilo kojem tipičnom protokolu za umrežavanje kojem je cilj da je put za podatke programiran. Međutim, pristup da je put za podatke programiran razlikuje se u *OpenFlow*-u. *OpenFlow* je spoj *client-server* tehnologije i raznih protokola umrežavanja. Može se ukratko objasniti sljedećim natuknicama:

- *OpenFlow* je primjer SDN-a i kao što ime sugerira, umrežavanje između dviju krajnjih točaka definirano je kroz *software* pokrenut na vanjskom računalu/poslužitelju, a radnje su programirane na *hardware*-u (*switch*-ima) na temelju inteligentne odluke kontrolera.
- *OpenFlow* je otvoreni protokol temeljen na standardima koji definira kako kontrolna cjelina može biti konfigurirana i kontrolirana sa centralnog mjesta (kontroler). Korištenjem *OpenFlow*-a, kontroler može upravljati kako će se paketi proslijediti kroz mrežu.
- U konvencionalnoj mreži, *switch*-evi i *router*-i imaju podatke pohranjene u različitim formatima (*routing* tablica, MAC tablica, itd.) koji se izračunaju složenim *switching* i *routing* protokolima koji su pokrenuti na cijeloj mreži ili skupu mreža. Na temelju tih tablica, podatkovna cjelina je programirana. *OpenFlow* protokol standardizira jedan jedini i centralizirani protokol koji može stvoriti tablice prosljeđivanja i upravljati njima, zamjenjujući sve ostale tablice prosljeđivanja. Podatkovna cjelina se potom programira na temelju tih tablica protoka.

*OpenFlow* i SDN se ponekad naizmjenično koriste u istoj rečenici zbog toga jer zajedno omogućuju SDN arhitekturu. Međutim, SDN je arhitektura, dok je *OpenFlow* protokol koji se koristi između kontrolera i mrežne infrastrukture kao što se vidi na slici 6.




Slika 6. Povezanost SDN arhitekture i *OpenFlow* protokola [7]

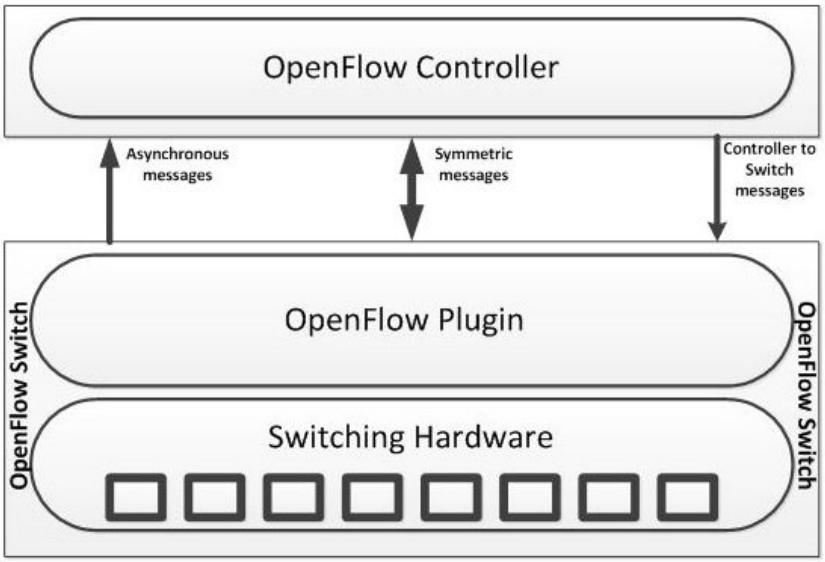
*OpenFlow* je dizajniran za centralno upravljanje i usmjeravanje mrežnog prometa za *router*-e i *switch*-eve, koji mogu biti od različitih proizvođača. Kao i bilo koji drugi jezik, *OpenFlow* protokol ima svoj riječnik koji se koristi između kontrolera i mrežne infrastrukture.

Sve poruke razmijenjene u *OpenFlow* protokolu mogu se podijeliti u tri glavne skupine (slika 7.):

- Poruke od kontrolera do *switch*-a,
- Asinkrone poruke te
- Simetrične poruke.

*Poruke od kontrolera do switch-a*: Te poruke su pokrenute od kontrolera i koriste se za upravljanje *switch*-eva ili da bi se dobile informacije od *switch*-eva te su podijeljene u šest vrsta poruka

- *Read State*: Koristi se od strane kontrolera za čitanje brojača koji se koriste za tokove, portove i za redove čekanja.
- *Modify State*: Prvenstveno se koristi za dodavanje, brisanje ili mijenjanje tokova u tablicama na *switch*-u.




Slika 7. Vrste poruka *OpenFlow* protokola [6]

- *Send packet*: Koristi se za slanje paketa iz određenog porta na *switch*-u.
- *Barrier Request/Replies*: Koristi se za primanje obavijesti završenih operacija *switch*-a. Ako je primljen zahtjev prepreke od kontrolera na *switch*-u, *switch* mora ispuniti sve zadatke na čekanju koje je imao prije zahtjeva. To stvara prepreku u *switch*-u tako da može otići do slijedećeg zadatka samo ako je završio trenutni zadatak. Čim je *switch* završio sve svoje zadatke, šalje odgovor prepreke natrag kontroleru i počinje raditi na svojim sljedećim zadacima.
- *Features*: Ovo je zahtjev mogućnosti prema *switch*-u nakon što je sigurna veza uspostavljena između kontrolera i *switch*-a. *Switch* odgovara samo kontroleru s mogućnostima i sposobnostima koje može podržati.
- *Configuration*: Koristi kontroler da bi primio i postavio konfiguracijske parametre u *switch*-u [8].

*Asinkrone poruke*: Te poruke daju *switch*-u priliku da slobodno komunicira s kontrolerom bez da traži dopuštenje. Ovo je mjesto gdje *switch* može obavijestiti kontroler o odbačenim paketima i sl. Postoje četiri različite vrste poruka:

- *Port Status*: Svaka promjena statusa porta poslana je kontroleru.
- *Packet in*: Može se činiti pomalo zbunjujućim, međutim ova poruka se šalje kontroleru samo ako postoji paket koji dolazi u *switch* i ne odgovara niti

jednom ulaznom toku (što znači da mora biti poslan kontroleru) ili paket odgovara pravilu da paket mora biti poslan kontroleru.

- *Flow removed*: Svaki ulazni tok dodan tablici protoka ima vrijednost mirujućeg isteka vremena i tvrdog isteka vremena povezanu s ulaznim tokom. Dakle, kad god je tok uklonjen, što može biti zbog mirujućeg isteka vremena, tvrdog isteka vremena ili poruke promjene toka, *switch* šalje *flow removed* poruku prema kontroleru. Poruka promjene toka određuje da li kontroler želi znati da je tok uklonjen ili ne.
- *Error*: Poruka se koristi kako bi *switch* obavijestio kontroler o greškama koje vidi [8].

*Simetrične poruke*: To su dvosmjerne poruke koje se međusobno šalju od strane kontrolera i *switch*-a. Postoje tri različite vrste poruka:

- *Hello*: Kao i mnogo drugih mrežnih protokola, hello poruke se razmjenjuju između *switch*-a i kontrolera na početku.
- *Echo*: Obje strane mogu pokrenuti ove vrste poruka i dužne su odgovoriti ako primaju poruku. Te poruke prikazuju kašnjenje između veze *switch*-a i kontrolera.
- *Vendor*: Poruke proizvođača su standardni način da *switch*-evi informiraju kontroler o dodatnim funkcijama koje oni nude. Ova poruka se također koristi za isprobavanje novih mogućnosti [36].

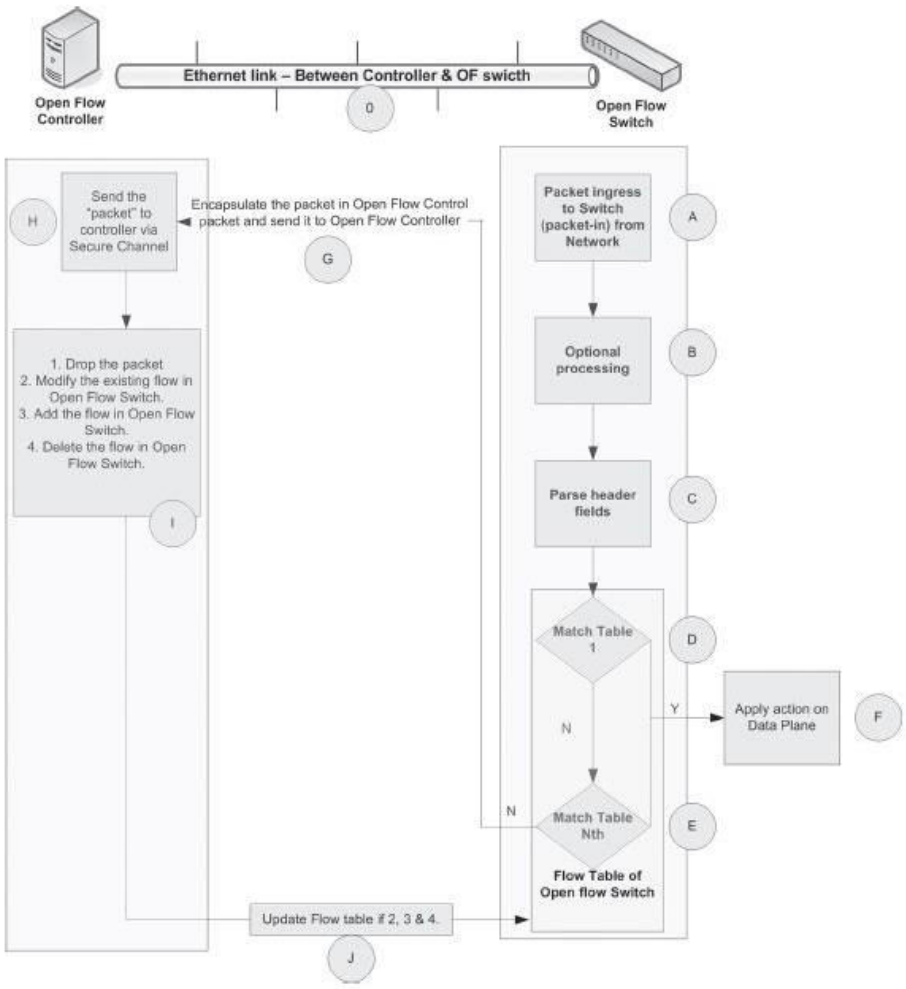
## 4.2. Pregled i komponente OpenFlow protokola

Ovo poglavlje objašnjava *OpenFlow* protokol s vrlo jednostavnim primjerom koji prikazuje što se događa kada paket dolazi do *OpenFlow switch*-a. Ovaj protokol obično radi na metodologiji klijent-poslužitelj, gdje poslužitelj programira klijenta (ili *switch*) te da im kaže kako da se ponašaju s protokom ili prometom. Najlakši način da se razumije rad *OpenFlow* operacije je da se prolazi kroz svaki korak dijagrama osnovnog toka.

Dijagram toka prikazan na slici 8. prikazuje dijagram osnovnog protoka te da bi bio osnovni protok, pretpostavlja se da je samo jedan link između *OpenFlow* klijenta (*switch*) i *OpenFlow* kontrolera (poslužitelj). Da bi se počelo na koraku 0,

trebala bi postojati veza između *OpenFlow switch*-a (koji je pokrenut u čistom *OpenFlow* modu) i *OpenFlow* kontrolera.

- Korak 0: Ovdje se uspostavlja veza između *OpenFlow switch*-a i *OpenFlow* kontrolera. Veza između kontrolera i jednog *switch*-a napravljena je jedan na jedan. Veza između poslužitelja i klijenta je TCP veza.
- Korak A: Ovaj korak objašnjava kada paket dolazi do *OpenFlow switch*-a, ako je dio *OpenFlow* mreže. U ovom dijagramu toka, pretpostavlja se da *OpenFlow switch* radi u *OpenFlow* modu i na taj način, kada paket dolazi do *switch*-a, predaje se do *OpenFlow* klijent modula koji je pokrenut u *switch*-u. Dolazni paket može biti kontrolni paket ili podatkovni paket.
- Korak B: U ovom koraku, *switch* se brine o razini obrade fizičkog sloja - implementacija specifična po proizvođaču i ovisi o vrsti linka kojeg *OpenFlow switch* ima. Nakon obrade paketa, predaje se *OpenFlow* klijentu koji se izvodi na *switch*-u klijenta.
- Korak C: Kada je paket unutar *switch*-a za obradu, *OpenFlow* klijent pokrenut unutar *switch*-a analizira zaglavlja paketa kako bi shvatio o kakvom se paketu radi.
- Korak D i E: Kada *switch* ima informaciju polja, počinje skeniranje tablice protoka, počevši od protoka 1 do protoka N. Kada je popis skeniran, uzima se radnja navedena u tom protoku. Različite implementacije mogu biti učinjene kako bi se optimiziralo skeniranje i kako bi tablica protoka bila učinkovitija, tako da broj skeniranja bude manji.
- Korak F: Ako polje odgovara, radnja povezana s tim specifičnim protokom će se obaviti. Tipična radnja je proslijediti paket, odbaciti paket ili ga poslati kontroleru. Ako je radnja proslijediti paket od *switch*-a, paket će biti preusmjeren. Ako nema navedene radnje za protok, onda je to implicitni pad za taj protok (kao takav, paket će biti odbačen). Ako nema ulaznih protoka koji se podudaraju s tim paketom, paket će biti poslan kontroleru (nakon što je enkapsuliran u *OpenFlow* kontrolni paket).
- Korak G i H: Ako se polja izdvojena iz paketa ne podudaraju s bilo kojom *entry-in-flow* tablicom, tada je paket enkapsuliran u *packet-in* zaglavlje i poslan kontroleru za daljnje radnje. Prema protokolu, kontroler mora odlučiti o tome što treba učiniti s novim protokom.




Slika 8. Dijagram toka *OpenFlow* protokola [1]

- Korak I: Kontroler pogleda paket i na temelju konfiguracije na kontroleru, odluči što će učiniti. Tipična odluka bila bi odbacivanje paketa, instalirati/modificirati tablicu protoka na *OpenFlow switch*-evima ili mijenjati postavke na samom kontroleru.
- Korak J: Ako se kontroler odluči za dodavanje/mijenjanje tablice protoka na *switch*-u, šalje odgovarajuću kontrolnu poruku *OpenFlow switch*-u. Nakon što je protok instaliran, sljedeći paket koji dolazi iz tablice protoka biti će zbrinut prema koraku F. Za svaki novi protok, ovi koraci će se ponoviti. Protoci mogu biti izbrisani na temelju implementacije na *OpenFlow switch*-u i *OpenFlow* kontroleru.

### 4.2.1. OpenFlow Controller

*OpenFlow* kontroler je jezgra svake *OpenFlow* bazirane softverski definirane mreže. To je *software* koji pokreće kontrolnu cjelinu i programira tokove na mrežnim *switch*-evima temeljenim na logici kontrolne cjeline (npr. najkraći putevi se izračunaju u kontroleru i zatim se programiraju na *switch*-evima). Kontroler se pokreće na snažnom poslužitelju, koji bi trebao biti dostupan svim *OpenFlow switch*-evima. Radi lakše upotrebljivosti, kontrole su dostupne administratoru/korisniku preko GUI-a, kako bi se upravljalo mrežom.

*OpenFlow switch*-evi će se obratiti kontroleru za svaki novi događaj koji nije naveden u tablici protoka. Trenutni kontroleri dostupni na tržištu imaju mogućnost dati cijeli pogled na mrežu na GUI-u te mnogi od njih mogu slikovito predstavljati tokove u mreži.

Drugi veliki razvojni segment SDN-a su aplikacije koje mrežni administrator može pisati na SDN kontrolerima. Trenutni kontroleri pružaju API-je, koji se mogu koristiti za razvoj aplikacija koje se mogu izvoditi na kontroleru i prilagoditi logiku kontrolne cjeline. Pisanjem aplikacija na kontrolerima, mrežni administratori mogu kontrolirati protoke.

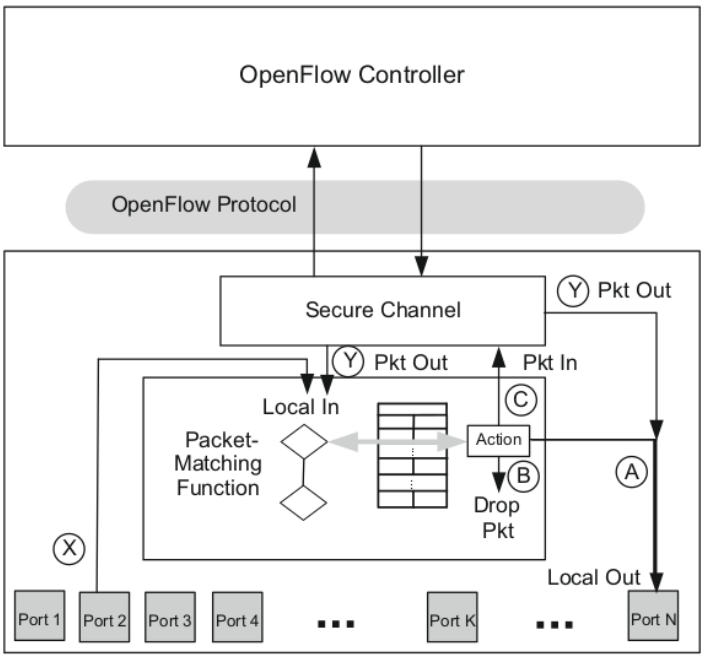
### 4.2.2. OpenFlow Switch

*OpenFlow switch* je tipičan *switch* umrežavanja koji će pokrenuti *software* potreban za spajanje na *OpenFlow* kontroler i instalirati i analizirati tokove. U ovoj fazi, definicija *OpenFlow switch*-a je specifična po dobavljaču/proizvođaču. Na visokoj razini, *OpenFlow switch* može biti svrstan u dvije kategorije:

1. *Pure OpenFlow Switch*: Podržava samo *OpenFlow* protokol
2. *Hybrid OpenFlow Switch*: Podržava dodatno uz protokole umrežavanja i konvencionalne *Ethernet* protokole [1]

Široka siva dvostruka strelica (koja se nalazi preko tablice protoka na slici 9.) počinje u logici odlučivanja, pokazuje podudaranje s određenim ulaskom u toj tablici i usmjerava paket koji se sada podudara prema *action box*-u na desnoj strani. *Action box* ima tri temeljne opcije za dolazeći paket:

- A: Prosljeđivanje paketa iz lokalnog porta, eventualno modificiranje određenih polja zaglavlja
- B: Odbacivanje paketa
- C: Puštanje paketa do kontrolera




Slika 9. *OpenFlow Switch* [7]

U slučaju C, paket je prošao do kontrolera preko sigurnog kanala koji je uglavnom osiguran TLS asimetričnom enkripcijom, iako su nekodirane TCP veze dopuštene. Ako kontroler želi slati kontrolnu poruku i podatkovni paket prema *switch*-u, kontroler koristi isti sigurni kanal u obrnutom smjeru. Kada kontroler želi proslijediti paket kroz *switch*, koristi *PACKET_OUT* poruku. Na slici se može vidjeti da takav paket (Y) koji dolazi iz kontrolera može ići različitim putevima [7].

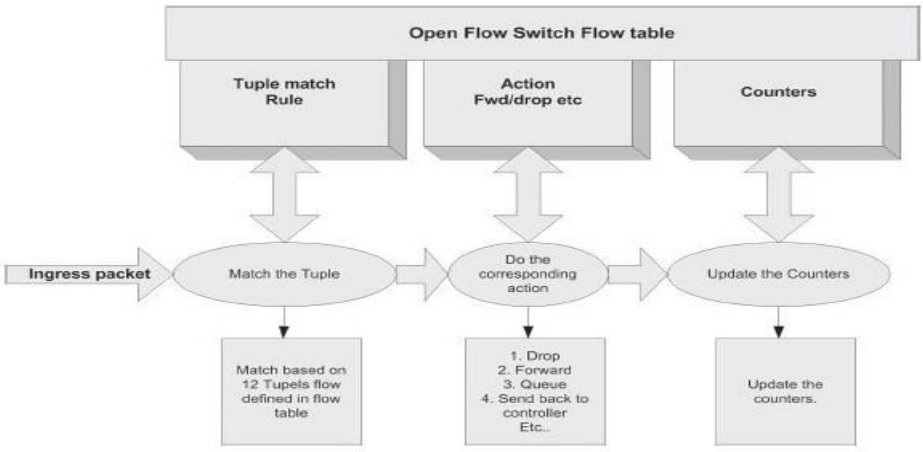
### 4.2.3. Flow Table

Tablica protoka (slika 10.) nalazi se u *OpenFlow switch*-u i govori *switch*-u kako postupati s bilo kojim protokom koji dolazi do tog *switch*-a (slika 11.). *OpenFlow* kontroler popunjava tablicu toka u *OpenFlow switch*-u i na temelju različitih tokova i fizičke topologije pogled samog kontrolera.

| Flow Entry 0 | | Flow Entry 1 | | | Flow Entry F | | | Flow Entry M | |
|---|---|---|---|---|---|---|---|---|---|
| Header Fields | Inport 12 192.32.10.1, Port 1012 | Header Fields | Inport * 209.*.*.*, Port * | ▪▪▪ | Header Fields | Inport 2 192.32.20.1, Port 995 | ▪▪▪ | Header Fields | Inport 2 192.32.30.1, Port 995 |
| Counters | val | Counters | val | | Counters | val | | Counters | val |
| Actions | val | Actions | val | | Actions | val | | Actions | val |

Slika 10. *Flow Table* [7]

Svaki dolazni promet u *OpenFlow switch*-u obrađuje se na temelju ove tablice protoka. Ako nema ulaza za određeni tok u tablici protoka, informacije protoka idu do kontrolera kako bi se odlučilo što učiniti s tim novim protokom koji dolazi do *switch*-a. Kada kontroler odlučuje kako obraditi novi protok, programira taj protok u *OpenFlow switch*-u s relevantnim radnjama.



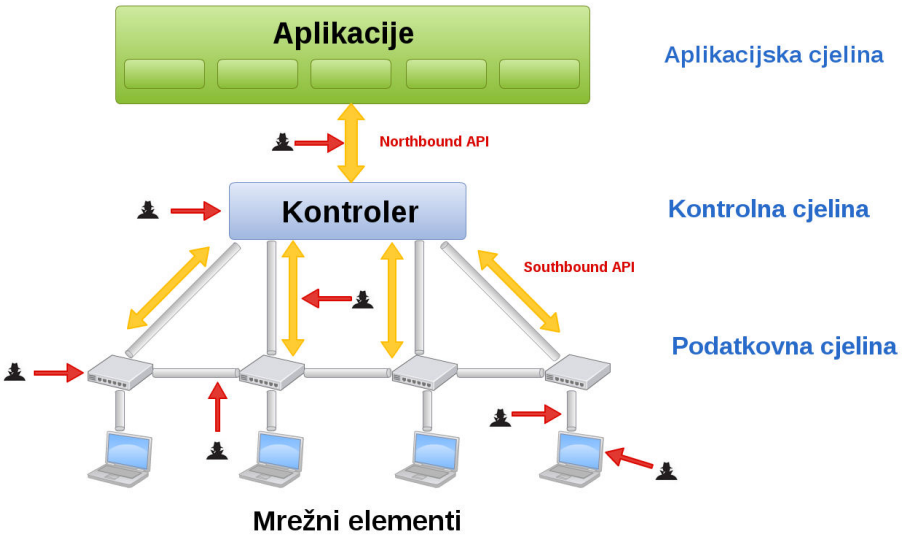

Slika 11. Postupak obrade protoka u tablici protoka [1]

Svaki ulaz tablice protoka uključuje:

- Polja zaglavlja koja se podudaraju s ulazećim paketima/protocima
- Brojači koji se ažuriraju za podudarajuće pakete/*byte*-ove po protoku
- Radnje koje bi se trebale poduzeti ako se protok podudara

# 5. SIGURNOST SOFTVERSKI DEFINIRANIH MREŽA

Sigurnost mreže jedan je od najvažnijih čimbenika u današnjim poduzećima, podatkovnim centrima i sl. i svakodnevno se ulaže puno truda u istraživanju novih prijetnji i novih vrsta napada. Stručnjaci zaduženi za sigurnost razvijaju nove metode obrane od postojećih, ali i novih vrsta napada kako bi osigurali poslovne informacije poduzeća ili informacije koje su skladištene u podatkovnim centrima. Do sada je provedeno stotina analiza i istraživanja vezanih za sigurnost mreže te mogući ciljevi napada i kako se obraniti od tih napada i u današnje vrijeme su poznati svim stručnjacima i mogu se također koristiti kod nekih elemenata SDN mreže. Međutim, zbog različite arhitekture SDN mreže u odnosu na konvencionalnu mrežu koja je prikazana u trećem poglavlju, postoje potpuno novi ciljevi napada u SDN mreži (slika 12.) i potrebni su novi mehanizmi osiguravanja mreže (engl. *network hardening*).




Slika 12. Mogući ciljevi napada unutar SDN arhitekture

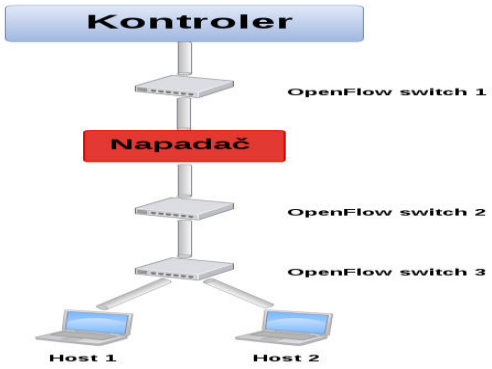
Izvor: Izradio autor

Sa slike se može uočiti da su ciljevi napada mogući u podatkovnoj cjelini gdje se mogu napast mrežni elementi i veza između njih te u samoj komunikaciji između SDN kontrolera i mrežnih elemenata, tj. preko *southbound* API-ja, u samoj kontrolnoj cjelini gdje se želi napasti tzv. "mozak" SDN mreže, tj. sami SDN kontroler te se može napasti u aplikacijskoj cjelini manipuliranjem samih aplikacija koji putem *northbound* API-ja komuniciraju s SDN kontrolerom. Prema tome su novi mogući

ciljevi napada u odnosu na konvencionalnu mrežu *southbound* i *northbound* API-ji te središte SDN mreže, sami SDN kontroler.

## 5.1. Ranjivost southbound API-ja

Kontroler komunicira s mrežnim elementom (*switch*) putem određenog protokola (*OpenFlow*, *OF-Config*, *OVSDB* i sl.). Svaki od tih protokola također ima svoje metode osiguravanja komunikacije prema mrežnim elementima. Nepotpuno znanje o tim metodama od strane npr. mrežnog administratora, dovodi do iskorištavanja tih slabosti od strane napadača. Jedan od mogućih načina napada može biti ubacivanje novih protoka u tablicu protoka samog mrežnog uređaja. Time želi lažirati nove protoke koje dozvoljavaju određenu vrstu prometa, koji bi zapravo trebao biti onemogućen unutar mreže. Ukoliko uspije stvoriti protok koji zaobilazi jedinicu za upravljanje prometa koja vodi promet do *Firewall*-a, napadač može imati veliku prednost. Ako može usmjeriti promet u željenom smjeru, to se može iskoristiti za njuškanje prometa i npr. izvođenje *Man in the Middle*[5] napada (slika 13.). Mogu se prisluškivati protoci kako bi se saznalo koji se protoci koriste, koja je vrsta prometa dopuštena unutar mreže, može se prisluškivati *southbound* komunikacija između mrežnog elementa i SDN kontrolera te dobivena informacija može biti korisna za ponavljanje napada ili se može iskoristiti u svrhu izviđanja.




Slika 13. *Man in the Middle* napad

Izvor: Izradio autor

[5] Man in the Middle: Man in the Middle je vrsta je napad koji se temelji presretanjem komunikacije između klijenta i poslužitelja te je jedna je od najčešćih tehnika dolaska do povjerljivih korisničkih informacija. Ubacivanjem u komunikacijski kanal uspostavljen između klijenta i poslužitelja napadač je u mogućnosti analizirati kompletni promet koji se razmjenjuje između ove dvije točke, čak i onda kada se koristi kriptirana komunikacija.

Prema dokumentu ONF-a koji navodi specifikacije *OpenFlow v1.0* [8], SDN kontroler i *switch* bi trebali komunicirati putem TLS[6] veze koja se pokreće kada kontroler prepoznaje sami *switch*. Autentifikacija se provodi razmjenom certifikata koji koriste privatne ključeve i time bi se trebalo spriječiti da napadači oponašaju *switch* ili kontroler te omogućuje šifriranje kontrolnog kanala kako bi se spriječilo prisluškivanje.

Korištenje TLS-a ima veću tehničku barijeru zbog koraka potrebnih da se ispravno konfigurira, što uključuje generiranje svestranih certifikata, generiranje certifikata za kontroler i za *switch*, potvrđivanje certifikata s ključevima te na kraju ispravno instaliranje tih certifikata i ključeva na svim uređajima. Zbog brzog razvoja *OpenFlow*-a, mnogo proizvođača *OpenFlow switch*-eva i kontrolera nisu u potpunosti implementirali specifikacije te su u potpunosti preskočili TLS. Tablica 1. prikazuje TLS podršku za popularne *OpenFlow switch*-eve i kontrolere. Može se uočiti da je podrška vrlo niska i razlog tome je nedovoljan interes.

Tablica 1. TLS podrške za popularne *OpenFlow switch*-eve i SDN kontrolere

| *Proizvođač switch-a* | *TLS podrška* | *Kontroler* | *TLS podrška* |
|---|---|---|---|
| HP | Ne | NOX | Da |
| Brocade | Ne | POX | Ne |
| Dell | Ne | Beacon | Ne |
| NEC | Djelomično | Floodlight | Ne |
| Indigo | Ne | Mul | Ne |
| Pica8 | Ne | FlowVisor | Ne |
| Open WRT | Da | Big Network Controller | Ne |
| Open vSwitch | Da | Open vSwitch Controller | Da |

Izvor: Izradio autor

Ne korištenje TLS-a dodatno vodi do sigurnosnih propusta samog protokola koji se koristi za komunikaciju između *switch*-a i kontrolera. Mogući propusti prikazani su u analizi [9]. Koristeći *Microsoft STRIDE* model za ispitivanje sigurnosnih prijetnji, koji uključuje *Spoofing*, *Tampering*, *Repudiation*, *Information disclosure*, *Denial of Service* (DoS) i *Elevation of privilege*, ispitala su se tri najčešće korištena protokola,

[6] TLS: Transport Layer Security i njegov prethodnik, Secure Sockets Layer (SSL), su kriptografski protokoli koji omogućuju sigurnu komunikaciju putem Interneta. TLS protokol dopušta aplikacijama komunikaciju putem mreže na način da sprječava prisluškivanje, uplitanje druge strane i krivotvorenje poruka. TLS omogućava autentičnost i privatnost komunikacije putem Interneta, budući da je šifriran.

*OpenFlow*, *OF-Config* i *OVSDB* i pronađeni su propusti za sva tri protokola, pogotovo ako se ne koristi TLS.

Nedostatak TLS podrške omogućuje napadačima da infiltriraju *OpenFlow* mreže i u tome uglavnom ostaju neotkriveni. Jednom kada se spoji između kontrolera i *switch*-a za presretanje *OpenFlow* prometa, može ubaciti vlastita pravila u samom *switch*-u kako bi snimao ili mijenjao promet, dobio pristup zaštićenim segmentima mreže, ometao pristup drugih uređaja te kontrolirao sve ostale *switch*-eve koji se nalaze ispod kontrolera.

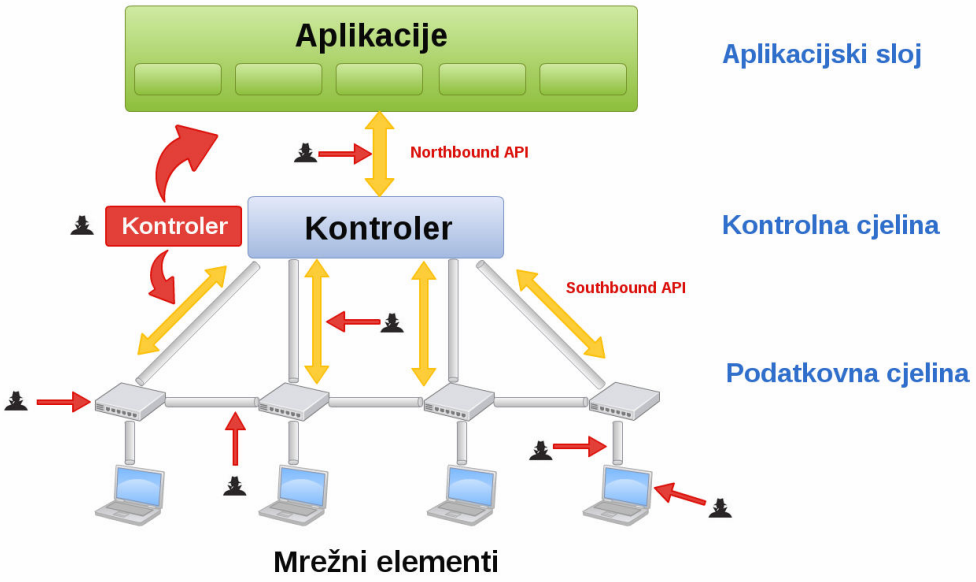
Posljedice koje predstavlja uspješan MITM napad u *OpenFlow* mreži nedvojbeno su gore nego u konvencionalnim mrežama zbog sposobnosti da napadač vrlo brzo može ponovno konfigurirati sve *switch*-eve koji se nalaze ispod. Za razliku, u konvencionalnoj mreži, napadač mora pričekati dok se npr. administrator prijavi na sučelje za upravljanje kod svakog *switch*-a posebno kako bi prikupio vjerodajnice. Kao što je prikazano na slici 13., napadač može presresti *OpenFlow* promet između kontrolera i *switch*-a te instalirati dodatno pravilo u *switch*-u 3, koje kopira npr. promet baze podataka između *host*-a 1 i *host*-a 2 te ga šalje prema udaljenom računalu. Kod konvencionalne mreže, zahtijeva se složenije ponovno konfiguriranje svakog *switch*-a nakon čekanja da se uhvate administrativne vjerodajnice.

## 5.2. Zaštita SDN mreže

Kao središte i tzv. mozak svake SDN mreže, očito je da će SDN kontroler biti glavna meta svakog napadača, jer je kontroler središnja točka koja utječe na rad mreže i koja vodi do neuspjeha cijele mreže. Zbog toga je važno da se vodi računa o nekim bitnim stvarima kao što su:

- Osiguranje kontrolera: Odvajanjem kontrolne i upravljačke cjeline, "mozak" je centraliziran, što teoretski omogućava da se rade promjene za poboljšanje brzine, učinkovitosti i za sigurnost mreže, sa samo nekoliko klikova. Budući da se kontroleri koji upravljaju mrežom mogu koristiti za veliki broj stvari, njihovo osiguranje od najveće je važnosti.

- Zaštita kontrolera: Zaštita dostupnosti kontrolera je također kritična. Poslovna rješenja moraju omogućiti smanjenje utjecaja koji jedan kontroler može imati za cijelu mrežu. Moguće je koristiti više kontrolera u jednoj mreži.
- Uspostava povjerenja između kontrolera, aplikacija i uređaja: Osigurati integritet bilo čega što komunicira s kontrolerom je kritični prvi korak u provjeri da li mreža radi kako treba. Mora biti jaka, uzajamna autentifikacija za aplikacije koje se pokreću na kontroleru, kao i za *switch*-eve, *router*-e i poslužitelje koje kontrolira. Također komunikacijski kanali moraju biti sigurni da se spriječi napad.
- *Policy Framework*: Potrebne su provjere kako bi se osiguralo da mreža radi kako treba. Kada su napravljene promjene na kontroleru, mora postojati *framework* kako bi se osiguralo da su promjene u skladu s korporativnim pravilima i ne otvaraju sigurnosne rizike.
- Forenzika i rješavanje problema: Kao u bilo kojoj mreži, razumijevanje što se događa ili što se dogodilo je bitno da se mogu napraviti promjene koje jačaju ukupnu sigurnost i kako bi zaštita bila bolja od budućih prijetnji.




Slika 14. SDN kontroler kao meta napada

Izvor: Izradio autor

Ukoliko je napad na kontroler prošao uspješno, napadač može ubaciti nove protoke. To može učiniti npr. podvaljivanjem *northbound* i *southbound* API poruka. Uspije li podvaliti protoke putem kontrolera, može dopustiti da promet teče cijelom

mrežom i najvjerojatnije, tj. sigurno će zaobići pravila koja se rabe za sigurnost mreže. Može se također izvesti DoS napad kako bi se uzrokovao neuspješan rad kontrolera, tj. pad kontrolera. Najgori scenariji je ako napadač stvori vlastiti kontroler i uspije da mrežni elementi vjeruju protocima lažnog kontrolera. Može stvoriti unose u tablici protoka mrežnih elemenata i SDN inženjeri neće moći vidjeti te protoke iz gledišta pravog kontrolera. Treba dodatno naglasiti da kontroleri često rade na *Linux* operativnom sustavu. Ako radi na operativnom sustavu opće namjene, sve ranjivosti tog operativnog sustava ujedno postanu i ranjivosti SDN kontrolera.

Osiguravanje sigurnosnog položaja kontrolera i mrežnih elemenata svodi se na osiguravanje OS-a *host*-a. Sve najbolje prakse za osiguravanje *public-facing Linux* poslužitelja mogu se ovdje upotrijebiti, kao što su:

- Šifriranje podatkovne komunikacije,
- Jedna mrežna usluga po sustavu ili VM instance,
- Fizička sigurnost sustava,
- Minimiziranje paketa/*software*-a za smanjenje ranjivosti,
- Ažuriranje *Linux kernel*-a[7] i *software*-a,
- Uključivanje *SELinux*-a,
- Omogućavanje *Firewall*-a,
- Odvojene particije te
- Korištenje centralizirane autentikacijske usluge i sl.

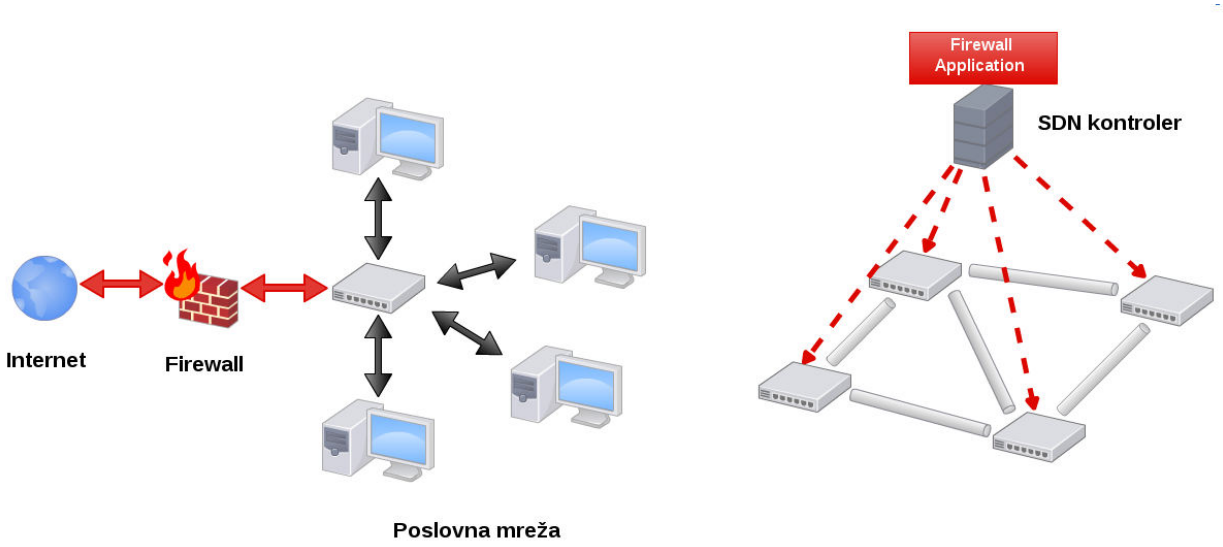
Dobar način osnovne zaštite samog kontrolera i cjelokupne mreže je korištenje *Firewall*-a čija se pravila direktno primjenjuju na samom kontroleru. Za razliku od konvencionalnog *Firewall*-a koji ne vidi unutarnji promet u samoj mreži te ga ne može filtrirati, SDN *Firewall* nadgleda sve u unutarnjoj mreži (slika 15.).

SDN *Firewall* je ujedno i filter paketa te ispitivač pravila. Prvi paket prolazi kroz kontroler i filtriran je od *Firewall*-a, sljedeći paketi protoka se direktno podudaraju s pravilima protoka. Arhitektura *Firewall*-a može biti:

- Centralizirana: Pravilo *Firewall*-a je definirano centralno i provodi se na kontroleru

[7] Linux kernel: Linux kernel je jezgra računalnog operativnog sustava. Linux operativni sustav se temelji na tome i razmještena je na tradicionalnim računalnim sustavima, kao što su osobna računala i poslužitelji.

- Distribuirana: Pravilo *Firewall*-a je definirano centralno, međutim propagira i provodi se na svakom pojedinom ulaznom protoku




Slika 15. Razlika između konvencionalnog i SDN *Firewall*-a

Izvor: Izradio autor

Pravilima *Firewall*-a se može upravljati fleksibilno od strane središnjeg servera, a SDN protokoli se mogu koristiti za standardno sučelje između *Firewall* aplikacija i *switch*-eva. Upotrebom SDN *Firewall*-a mogu se ujedno i očekivati manji troškovi, jer je jedan *Firewall* dovoljan, može se adaptivno postaviti ovisno o uvjetima u mreži, pravila *Firewall*-a se mogu dinamično dodati za vrijeme novih napada, što olakšava samo upravljanje *Firewall*-om te centralni pogled može biti koristan za određivanje sigurnosnih pravila.

*Northbound* API-ji su također meta napada, pošto direktno komuniciraju s kontrolerom i napadač ima veći izbor kako će točno napasti. *Northbound* API-ji mogu koristiti *Java*-u, *C*, *JSON*, *Python*, *REST* i *XML*. Iskorištavajući njihove slabosti napadač može steći kontrolu nad SDN infrastrukturom. Često za takve API-je postoji zadana lozinka i ukoliko se ta lozinka ne promijeni, napadač ju može lako pogoditi. To mu omogućava da stvori pakete koji će se proslijediti sučelju za upravljanje na kontroleru kako bi utvrdio strukturu SDN mreže i ubacio svoje vlastite konfiguracije. Korištenje TLS-a, SSH-a ili drugog načina osiguranja *northbound* komunikacija i osiguranja upravljanja kontrolera, može se smatrati najboljom praksom. Komunikacije od strane aplikacija i usluga koje traže usluge ili podatke od kontrolera, trebale bi biti

osigurane te se treba koristiti provjera autentičnosti i metode šifriranja. Sigurne prakse kodiranja za sve *northbound* aplikacije koje traže SDN resurse također se smatraju najboljom praksom. Nisu samo sigurne prakse kodiranja korisne za sigurnost *public-facing* Internet web aplikacija, također se mogu primijeniti za *northbound* SDN veze. SDN također omogućuje razvoj sigurnosnih aplikacija za spriječavanje ili ublaživanje napada, koje koriste centraliziranu kontrolnu arhitekturu SDN-a.

Nekoliko istraživanja bavila su se sigurnosnom analizom SDN-a, posebno *OpenFlow*-a. [10] daje pregled ranjivosti uzrokovane odvajanjem kontrolne i podatkovne cjeline. [11] pokazuje prijetnje SDN-a i predlaže dizajn za siguran i pouzdan SDN. Formalna sigurnosna analiza SDN arhitektura obavljena je od strane *Internet Draft SDN Security Requirements* [12]. [13] prikazuje sigurnosnu analizu *OpenFlow*-a i predlaže tehnike sprječavanja i ublažavanja. [14] spominje izazove implementacijom SDN-a s naglaskom na performanse mreže, skalabilnost, sigurnost i interoperabilnost te predlaže moguća rješenja. Za rješavanje i ublažavanje problema SDN dizajna, proširenje podatkovne cjeline, *AVANT-GUARD* [15] smanjuje *data-to-control plane* komunikaciju za ublažavanje DDoS napada. [16] predstavlja sustav dopuštenja *PermOF*, koji ima za cilj dati *OpenFlow* aplikacijama minimalne dozvole za sprječavanje zlouporabe. [17], [18], [19] predstavljaju *FleXam*, proširenje za pristup informacijama o razini paketa od strane kontrolera za npr. klasifikaciju prometa. [20] uvodi *framework CloudWatcher* za praćenje velikih i dinamičnih cloud mreža. [21] pokazuje metodu za otkrivanje DDoS napada na temelju značajki protoka prikupljenih u SDN mreži. [22] pokazuje postupak napadača za *fingerprint* SDN-a u pripremi za DoS napad. [23] uvodi sustav modela provjere *Flover*, koji potvrđuje da *OpenFlow* pravila ne krše sigurnosna pravila mreže. [24] i [25] uvode *Framework for Enabling Security Controls* (FRESCO) u *OpenFlow* mrežama za integraciju sigurnosnih aplikacija u SDN-u.

Može se samo pokušati predvidjeti na temelju SDN arhitekture gdje napadači žele ciljano napasti mrežu. Implementacije su nove, kao i protokoli i *software* kontrolera te je povijest posljednjih SDN napada dosta nepoznata. Prije nego što jedna organizacija kreće u projekt SDN mreže, treba razmisliti kako će osigurati sustav u ranoj fazi samog projektiranja i ne smije se čekati dok cijela mreža već radi.

Kao i većina stvari, postavljanje i konfiguracija sigurnosti odmah od početka će uštedjeti organizacijama mnoge probleme.

Na temelju brojnih istraživanja i analiza može se zaključiti da SDN uz odgovarajuće mehanizme može pojačati sigurnost mreže. S jednog mjesta se može nadgledati cijela mreža, kao što su aplikacije, podaci, identitet korisnika i identitet uređaja te ukupno ponašanje mreže. Alati za analizu će iskoristiti tu informaciju koja dolazi iz svih uređaja u mreži, ne samo od sigurnosnih uređaja, da se pronađu prijetnje i da se bolje reagira na njih. S SDN sigurnosnim aplikacijama, može se imati daleko brža sigurnost te se na kraju mogu bolje zaustaviti prijetnje.

Budući da kontroler ima jedinstveni pogled na mrežu, lakše se može odrediti gdje se *Firewall* uređaji nalaze. Jedna kontrolna točka može osigurati da je sav promet preusmjeren na *Firewall*. Svaki protok se podudara s skupom pravila, jer se ne moraju instalirati konfiguracije i pravila na mnoštvo uređaja. Nadogradnje za *anti-malware* programe mogu biti puno lakše. Pravila se mogu definirati na razini mreže, umjesto na razini *Firewall*-a.

Kontrola u stvarnom vremenu znači da se može brzo djelovati na naprednim prijetnjama, bez obzira gdje se nalaze u mreži. Nakon što se mogu "vidjeti" prijetnje u mreži, djelovanje prema prijetnjama je puno brže. Koristeći inteligenciju sigurnosnih uređaja za rad u kombinaciji sa širokim skupom mrežnih uređaja omogućiti će da se blokira promet, odbace zaražena računala i spriječi napredni *malware* i sl.

# 6. PLANIRANJE SOFTVERSKI DEFINIRANIH MREŽA

Mnoge organizacije pojačavaju inicijativu implementacije na SDN rješenje, međutim postavlja se pitanje kako najbolje provesti prijelaz na više automatiziranu mrežnu arhitekturu te o čemu treba razmisliti i koji su postupci.

Organizacije trebaju imati jasnu ideju o prednostima za koje misle da će ih realizirati implementacijom SDN-a. U mnogim slučajevima, softverski definirano rješenje ne mora nužno izgledati drugačije od konvencionalne mreže. Informatički tim mora odrediti utjecaj SDN modela na postojećim uslugama. Trebali bi raditi s uzorcima aplikacija koje će se koristiti za povezivanje i provjere kontinuiteta usluga prije i poslije implementacije. To će pomoći ne samo protiv prekida usluga, nego će i ukloniti sve probleme koje su povezane s implementacijom. No, bez obzira na to koliko se priprema provodi, prije toga se ne mogu predvidjeti sve okolnosti. Zbog tog razloga, bitno je da postoji plan koji omogućuje administratorima povratak na prethodnu konfiguraciju mreže.

Nema sumnje da implementacija SDN-a bez adekvatnog znanja predstavlja određeni rizik, međutim ignoriranje SDN-a predstavlja značajan rizik i za IT organizacije i za IT stručnjake. Rizik za IT organizacije je taj da neće biti u stanju riješiti probleme za koje je SDN dizajniran, što rezultira nedostatkom u konkurentnosti. Rizik za IT profesionalce je taj da će zakasniti s učenjem i samom edukacijom koja vezana za ovaj pristup i time neće biti konkurentni trenutnom ili budućem poslodavcu.

Prema raznim analizama i istraživanjima se može zaključiti da će SDN u narednih nekoliko godina imati značajan utjecaj i na mreže poduzeća i na uloge samih mrežnih stručnjaka. Zbog toga IT organizacije i IT stručnjaci moraju razviti plan za implementaciju SDN-a. S obzirom na brzo mijenjanje samih tehnologija, pa tako i SDN-a, bilo koji plan implementacije će se dalje razvijati s vremenom.

U literaturi [26] opisan je proces razvijen od stručnjaka za analizu i istraživanje SDN mreža, koji se može uzeti u obzir od jedne organizacije za procjenu rješenja SDN-a i eventualno provođenje jednog ili više rješenja te uključuje sljedeće korake:

1. Definiranje SDN-a,
2. Prepoznavanje primarnih mogućnosti,
3. Prepoznavanje ključnih metrika,
4. Odlučivanje,
5. Procjena SDN rješenja,
6. Ispitivanje i certificiranje rješenja,
7. Integracija s postojećom okolinom,
8. Educiranje organizacije,
9. Ocjenjivanje profesionalnih usluga,
10. Uklanjanje organizacijskog otpora,
11. Izvršavanje POC-a (*proof of concept*) te
12. Dobivanje *Management Buy-In.*

Implementacija se može razlikovati ovisno o veličini i složenosti mreže te iskustvu IT tima. Potrebne su nove vještine i dodatna obuka. Ipak, prijelaz na arhitekturu koja obuhvaća *software* nije toliko težak koliko se pretpostavlja. Uz pravilno planiranje, većina organizacija može brzo i jednostavno iskoristiti sve prednosti SDN rješenja.

S obzirom na razvijanje SDN-a i njegovih koncepata, postoje određeni zahtjevi i izazovi za upravljanje SDN mrežom. Razmotrit će se neki od najvažnijih zahtjeva i izazova s obzirom na sve cjeline SDN arhitekture.

*Pokretanje sustava i konfiguracija*

U konvencionalnim mrežama logika cjeline prosljeđivanja i kontrolne cjeline ograničena je unutar svakog mrežnog uređaja u skladu s dobro definiranim skupom standardiziranih protokola. Razdvajanjem cjelina, potreba za pokretanjem komunikacije između cjeline prosljeđivanja i kontrolne cjeline postaje osnovni uvjet. Konfiguriranje ove komunikacije može biti složena, s obzirom da obje cjeline mogu djelovati pod protokolima definiranih *software*-om. Štoviše, promjene *software*-a u bilo kojoj cjelini mogu izravno utjecati na takvu komunikaciju. U idealnom slučaju, novi softverski definirani protokoli bi trebali sadržavati sučelje za upravljanje kako bi se omogućilo pravilno pokretanje i konfiguriranje u tim mrežama.

*Dostupnost i otpornost*

Kao i u bilo kojoj drugoj mreži, SDN je osjetljiv na greške u fizičkim ili logičkim elementima, primjerice, kvarovi u linkovima ili softverske pogreške mreže. Posebno, s razdvajanjem cjelina, postoji mogućnost eventualnog prekida veze između cjeline proslijeđivanja i kontrolne cjeline. U mnogim se pristupima SDN-a čak i razmotri stavljanje cijele upravljačke cjeline unutar jednog čvora, međutim u takvom slučaju pitanja nedostupnosti i otpornosti postaju još kritičnija zbog jedne točke neuspjeha (engl. *single point of failure*). Unatoč činjenici da su uređaji za prosljeđivanje obično u stanju reagirati na ovaj problem, još uvijek je važno upravljati vezom između cjelina i da li je aktivna veza, u skladu s pravilima mreže.

*Mrežno programiranje*

U svakodnevici SDN administratora, svako novo puštanje mrežnog *software*-a ili ažuriranje se mora također konstantno primijeniti i na implementacijama cjeline prosljeđivanja i kontrolne cjeline preko mreže. Da bi se osigurala odgovarajuća podrška za upravljanje mrežnim programiranjem, potrebni su alati koji omogućuju mrežnim administratorima kontrolu verzije, koordiniranu implementaciju, povrat na početno stanje i provjera mrežnog *software*-a. Štoviše, *software* za diktiranje ponašanja mreže može se također razvijati u različitim razinama apstrakcije (visoka ili aplikacijska razina, srednja razina ili razina kontrolne cjeline i niža razina ili razina cjeline prosljeđivanja). Alati i metode trebaju biti dizajnirani da rastavljaju ova pravila ili naredbe visoke/srednje razine dolje prema konfiguraciji uređaja niske razine.

*Performanse i skalabilnost*

U SDN-u, mrežni *hardware* mora biti više generički da bi *software* mogao biti napisan na visokim razinama apstrakcije. Štoviše, odvajanje samih cjelina podrazumijeva dodatne komunikacijske zahtjeve između cjelina, koji mogu dodati kašnjenja mrežnom prometu koji se prosljeđuje. Dio odgovornosti za osiguranje učinkovitosti pripada razvijateljima *software*-a koji moraju dizajnirati optimiziran *software* za mreže. Drugi dio pripada mrežnim administratorima koji moraju uskladiti ispravne parametre za optimizaciju softverski definiranih protokola te odabrati najučinkovitije kontrolne i upravljačke modele koji će se koriste (npr. centralizirane, distribuirane ili hijerarhijske) [27].

Kako postići uspješan prijelaz na SDN rješenje bavili su se autori u literaturi [14]. Specifičan fokus njihove analize bio je na izazove vezane za performanse, skalabilnost, sigurnost i interoperabilnosti mreže te predlažu određena rješenja.

*Izolacija i sigurnost*

U SDN-u se ne dijeli samo mrežni promet među mnogim korisnicima i aplikacijama, nego je i mrežna logika sama po sebi kontrolirana iz daljine putem prilagođenog *software*-a. Nove tehnike izolacije resursa i sigurnosnog upravljanja se moraju istražiti da bi se jamčila izolacija u svim cjelinama SDN-a kao i u njihovoj komunikaciji. Naime, mora se osigurati da se kontrolni promet (protoci između kontrolera i uređaja prosljeđivanja) izolira i taj kod napisan od jednog programera ne utječe na kod napisan od strane drugih. Kako SDN postaje sve popularniji, ranjivosti će se pojaviti i uskoro će se početi pisati prvi zlonamjerni kodovi za te mreže.

*Fleksibilnost i razdvajanje*

Sučelja su potrebna kako bi se omogućila razmjena informacija upravljanja iz/do cjeline prosljeđivanja, kontrolne i aplikacijske cjeline. Budući da se ponašanje mreže sada može definirati *software*-om, upravljanje SDN-om treba biti dovoljno fleksibilno za brzu prilagodbu novim protokolima pisanih za sve cjeline.

*Mrežno planiranje*

Tradicionalno, administratori trebaju planirati i implementaciju i proširenje mreže, što obuhvaća zadatke kao što su definiranje kapaciteta i performansnih potreba te odlučivanje hoće li i gdje u topologiji mreža biti segmentirana (na slojevima 2 i 3). Ove odluke će rezultirati skupom mrežnih *hardware* kutija (npr. *router*-i, *switch*-evi i *Firewall*-i) koji rade pod dobro poznatim standardnim protokolima. Planiranje nije uvjet za upravljačku cjelinu SDN-a po sebi, ali utječe na ostale zahtjeve. U SDN-u, administratori sada trebaju steći skup uređaja za prosljeđivanje i eventualno još jedan set kontrolnih uređaja. Na vrhu toga, topologija mreže i planiranje kapaciteta može biti pod utjecajem vrste *software*-a koji je instaliran. Administratori su još uvijek u mogućnosti birati između različitih dobavljača mrežnog *software*-a, kontrolnih uređaja i uređaja za prosljeđivanje. Postavljanje ovih elemenata preko mrežne topologije može izravno utjecati na performanse i otpornost mreže. Osim toga, upravljačka

cjelina može pomoći administratoru u izradi odluka planiranja kroz niz alata za planiranje raspoređenih u sučelju za upravljanje.

*Nadzor i vizualizacija*

Sa fizičke strane, zahtjevi nadzora i vizualizacija ostaju slični konvencionalnim mrežama. Logički dio je daleko složeniji pod pretpostavkom da se protokoli prosljeđivanja i kontrolni protokoli mogu u potpunosti redizajnirati. Npr. u trenutnim IP mrežama lako je nacrtati kartu dostupnosti analizirajući podatke *routing* tablica, čak i prije nego promet počinje teći. Takva analiza je moguća samo zato što je ponašanje prosljeđivanja IP *router*-a predvidljivo. Kada unutarnja implementacija jednog protokola nije unaprijed poznata, odnosno definirana *software*-om, to više nije moguće.

*Pravila visoke razine i mrežna konfiguracija*

SDN cjeline niže razine predstavljaju vlastitu apstrakciju *software*-a i izlažu API-je za obradu tih apstrakcija do cjelina veće razine. Svaki put kada je razina apstrakcije podignuta, postoji gubitak podataka jer nisu sve pojedinosti niže razine dostupni kroz definirane API-je. Kada pravila ili naredbe vrlo visoke razine dolaze iz najviših cjelina SDN arhitekture, izgubljena informacija se treba rekonstruirati ili prevesti u skup radnji niže razine. Za pravilno rješavanje pokretanja i konfiguracijskih zahtijeva, postupci za prevođenje pravila visoke razine u konfiguracije niske razine postaje izazov. Iako veliki broj literature za upravljanje mrežom na temelju pravila predstavlja solidnu osnovu u tom smislu, i dalje je SDN usmjereno istraživanje potrebno za rješavanje ovog izazova [27].

*Autonomno upravljanje i upravljanje unutar mreže*

U idealnom slučaju, cjelina upravljanja se treba usredotočiti na sve funkcije upravljanja mrežom SDN-a. Ipak, budući da SDN omogućuje *software*-u da diktira ponašanje mrežnih uređaja, trebalo bi biti moguće smjestiti kod unutar tih uređaja da bi im se omogućilo da reagiraju na mrežne uvjete i obavljali funkcionalnosti samostalnog upravljanja. Pristupi autonomnog upravljanja i upravljanja unutar mreže nude mogućnost migracije upravljačkih funkcija, obično raspoređenih u cjelini upravljanja, da se *software* pokreće na uređajima cjeline prosljeđivanja i na uređajima kontrolne cjeline. Ovi pristupi su također važni jer pomažu rješavanju

zahtijeva za dostupnost i otpornost u SDN-u. Ovi pristupi općenito se smatraju u situacijama u kojima su količina uređaja ili uvjeti mrežnog povezivanja neprikladni za održavanje učestale komunikacije između uređaja i cjeline upravljanja.

*Fleksibilno upravljanje putem sučelja*

U SDN arhitekturi, cjelina upravljanja je u interakciji s drugim cjelinama putem korištenja sučelja. Odgovarajuća sučelja za upravljanje su također potrebna, npr. kako bi se omogućio više integrirani pogled mrežnih resursa s sustavom pokrenut na vrhu, kao što je u *Cloud* okruženjima. Kako se nositi s zahtjevom fleksibilnosti i razdvajanjem, izazovom pronalaženja sustavnog načina definiranja, dovodi do korištenja i razvijanja sučelja za upravljanje.

*Pametno planiranje mreže*

Administratori moraju uzeti u obzir tri glavna aspekta kada planiraju mrežnu infrastrukturu:

1. Fizičko razdvajanje cjelina
2. Apstrakcije *software*-a koje pokreću logiku mreže
3. Postojanje jednog ili više kontrolnih entiteta

Što se tiče aspekata 1 i 3, temeljno je da se uzima u obzir da broj elemenata u svakoj cjelini može biti proizvoljan. Štoviše, s obzirom na aspekt 2, odabir pravih aplikacija za mreže postaje bitan i netrivijalan zadatak. Osim kao jedan uvjet u SDN-u, planiranje je također i izazov, jer ako je virtualizacija upotrebljena u SDN-u da bi se podržale različite mreže, količina informacija s kojom mrežni administrator treba rukovati je mnogo veća u usporedbi s mrežom koja nije virtualizirana [28]. Pametna rješenja za planiranje postaju neophodna kako bi se pomoglo mrežnim administratorima s pitanjima prije korištenja SDN-a. To mogu biti pitanja kao što su:

- Koliko upravljačkih i kontrolnih uređaja je potrebno za razvijanje mreže?
- Gdje točno trebaju biti fizički postavljeni kontrolni uređaji u topologiji u odnosu na one za prosljeđivanje?
- Kako su kontrolne i upravljačke odgovornosti organizirani (npr. centralizirano, distribuirano ili hijerarhijski)?
- Kako izbor *software*-a utječe na fizičku topologiju i sl.?

*Situacijsko upravljanje*

U SDN mreži su uvjeti mnogo više dinamični zbog lakših i čestih instalacija i promjena *software*-a. Kao posljedica toga, vjerojatno će se stvoriti situacijski problemi, koji nisu predviđeni u dizajnu sustava upravljanja. Za suočavanje s neočekivanim i privremenim problemima, kao što je ispravljanje pogrešaka nedavno instaliranog softverskog protokola, alati moraju biti dostupni za brzu izradu *on-demand* aplikacija za upravljanje iskorištavanjem dostupnih informacija iz SDN cjelina i sučelja. Adekvatno situacijsko upravljanje obuhvaća ispunjavanje zahtjeva za nadzor i vizualizaciju SDN-a [28].

Prema trenutnoj analizi SDN-a u ovom radu, uključujući zahtjeve i izazove, uspješna implementacija SDN-a može imati sljedeće prednosti u odnosu na konvencionalnu mrežu:

- Agilnost i fleksibilnost u upravljanju
    - SDN omogućuje organizacijama brzi razvoj novih aplikacija, usluga i infrastruktura kako bi se brzo zadovoljili promjenjivi poslovni ciljevi
    - fleksibilan odabir načina i djelovanja mreže
    - omogućuje IT menadžerima da eksperimentiraju s mrežnom konfiguracijom bez da utječu na samu mrežu
    - sposobnost oblikovanja i kontrole podatkovnog prometa, koja je jedna od glavnih prednosti SDN-a
    - zbog centraliziranog pogleda na cijelu mrežu omogućeno je i samo upravljanje mreže s jedne točke
    - podržava upravljanje fizičkih *switch*-eva, virtualnih *switch*-eva i mrežnih uređaja sa jednog centralnog kontrolera
    - pruža jedan set API-ja za kreiranje upravljačke konzole za fizičke i virtualne uređaje
    - moguće korištenje proizvoda od više proizvođača
- Smanjenje troškova
    - administrativne učinkovitosti, poboljšanje iskorištenosti poslužitelja, bolja kontrola virtualizacije i druge pogodnosti trebale bi rezultirati operativnim uštedama

    - ne zahtijeva velika ulaganja jer nema potrebe za skupim mrežnim uređajima
    - zbog mogućnosti korištenja virtualnih *switch*-eva, može se više *switch*-eva koristiti na jednom *hardware*-u
    - nekoliko SDN proizvoda su besplatni
- Bolja sigurnost
    - SDN kontroler pruža središnju točku za distribuciju sigurnosnih informacija i pravila kroz cijelu mrežu
    - SDN kontroler nadgleda cijelu mrežu te se može brzo reagirati ukoliko se primjećuju nepravilna ponašanja mreže
    - iako je središnja točka napada, uz sigurnu i pravilnu implementaciju, može se učinkovito upravljati sigurnošću mrežom

Navedene prednosti se odnose prema dosadašnjoj analizi u ovom radu te se može reći da su osnovne prednosti za bilo koju vrstu mreže. Ovisno o vrsti mreže, usluga koje pruža i drugih faktora, postoje još prednosti koje se razlikuju od mreže do mreže.

# 7. PRIMJENA I SIMULACIJSKI PRIKAZ RADA SOFTVERSKI DEFINIRANE MREŽE

U ovom poglavlju će se prikazati rad jedne SDN mreže s *OpenFlow switch*-evima i kontrolerom. Konfiguracija SDN mreže će se usporediti s konfiguracijom jedne konvencionalne mreže kako bi se uočile bitne razlike samih konfiguracija.

Za izradu mrežne topologije konvencionalne mreže i za konfiguraciju *switch*-eva koristiti će se *Graphical Network Simulator* 3[8] (GNS3) instaliran na *Linux* OS-u, koji omogućuje kombinaciju virtualnih i stvarnih uređaja te omogućuje simulaciju kompleksnih mreža. Koristi *Dynamips*[9] emulacijski *software* za simulaciju Cisco *Internetwork Operating System* (Cisco IOS). *Software* korišten pri konfiguraciji *switch*-eva se koristi i u stvarnim fizičkim uređajima te prikaz konfiguracije u nastavku rada odgovara pravim fizičkim *switch*-evima.

Izrada SDN mrežne topologije kako bi se prikazao rad kontrolera obavit će se pomoću *Mininet*[10] emulatora, koji omogućuje kreiranje realnih virtualnih mreža i pokreće stvarni *kernel*, *switch* i aplikacijski kod na jednoj virtualnoj mašini, u ovom slučaju na *VM VirtualBox*-u. SDN kontroler koji će se koristiti je *OpenDaylight Controller*[11].

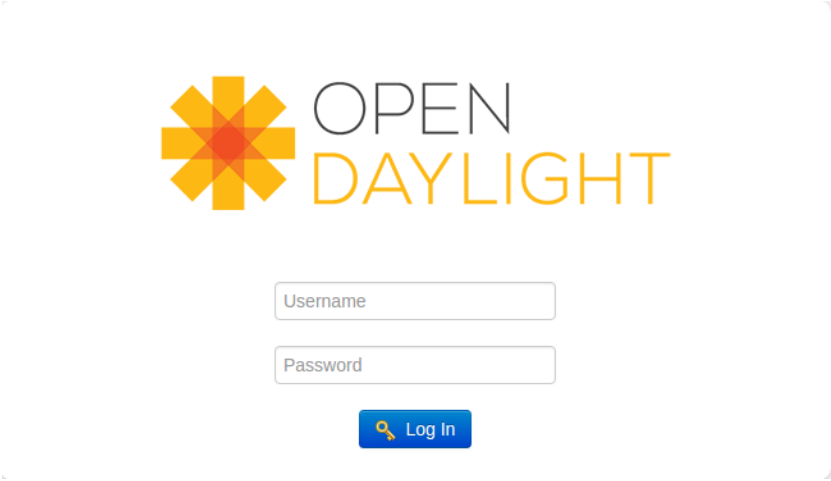
## 7.1. OpenDaylight Controller

Nakon određenog postupka instalacije kontrolera i njegovog pokretanja, pristup se može omogućiti unutar web preglednika unosom *loopback* IP adrese *localhost*-a i portom 8080, koja odgovara adresi 127.0.0.1:8080. Pojaviti će se stranica za prijavu na kontroler, što je prikazano slikom 16. Tvornički *username* i *password* su admin. Nakon prijave se korisnici i lozinke mogu promijeniti.

Uspješnom prijavom otvara se GUI *OpenDaylight* kontrolera (slika 17.) i kao što je vidljivo nema povezanih mrežnih elemenata. Zbog nemogućnosti prikaza

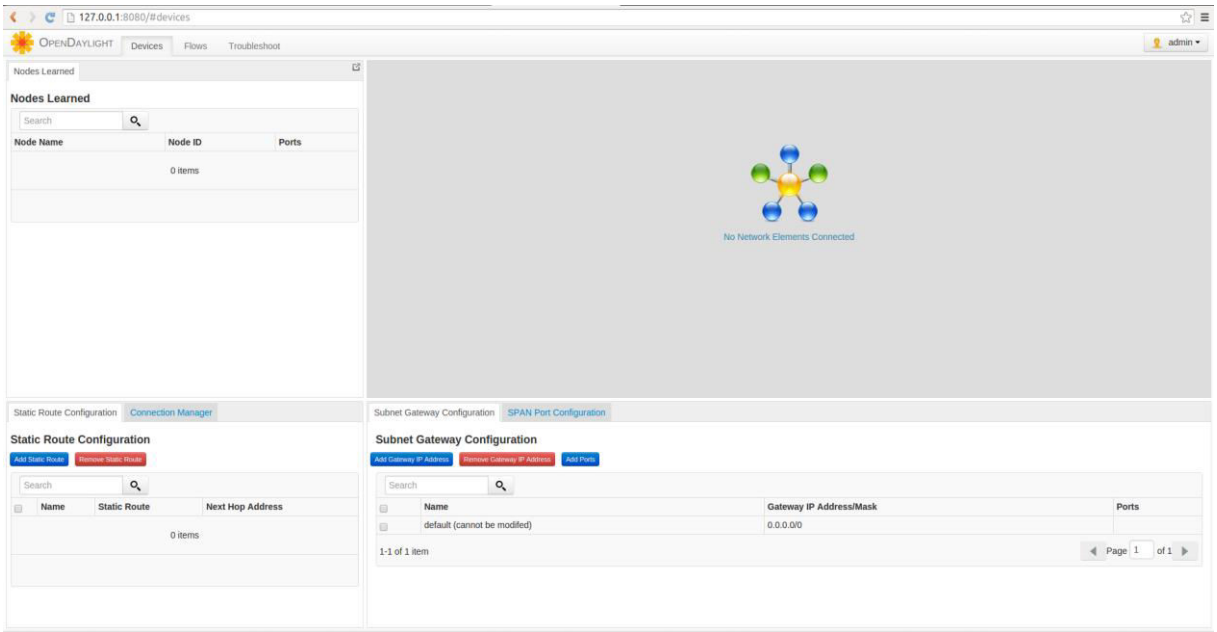
[8] Graphical Network Simulator 3: http://www.gns3.com
[9] Dynamips: Dynamips je računalni emulator program koji je napisan da oponaša Cisco router-e. Dynamips radi na FreeBSD, Linux, Mac OS X ili Windows i može oponašati hardware Cisco router platformi izravno pokretanjem stvarnog Cisco IOS software image-a u emulatoru.
[10] Mininet emulator: http://www.mininet.org
[11] OpenDaylight: http://www.opendaylight.org

funkcija kontrolera, ukoliko nema povezanih mrežnih elemenata, koristiti će se topologija čija će konfiguracija biti objašnjena u poglavlju 7.3.




Slika 16. Prijava na *OpenDaylight* kontroler

Izvor: Izradio autor

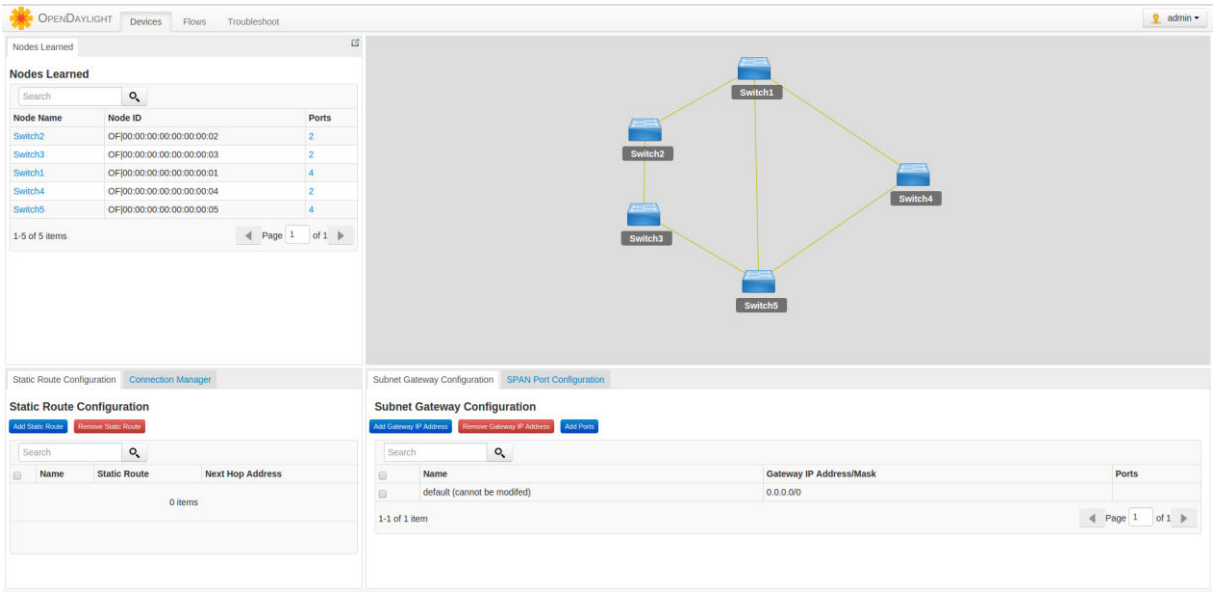



Slika 17. GUI *OpenDaylight* kontrolera

Izvor: Izradio autor

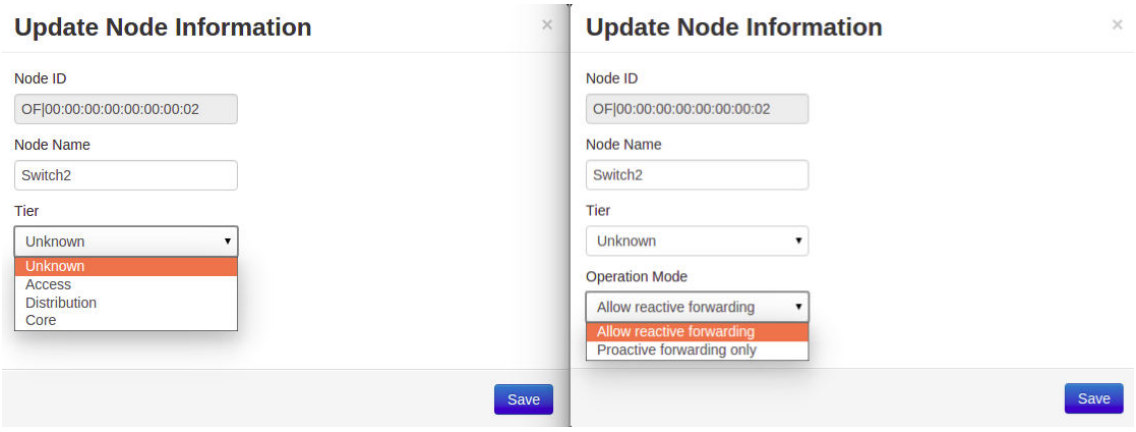
Ukoliko je povezivanje mrežnih elemenata bilo uspješno, kontroler bi trebao vidjeti sve *switch*-eve te je izgled GUI-a prikazan na slici 18. Na vrhu stranice se nalazi izbornik i može se izabrati *Devices*, *Flows* i *Troubleshoot*. Odabirom *Devices*

prikazuje se stranica kao na slici 18. S lijeve strane su prikazani naučeni čvorovi zajedno s ID-om (MAC adresa) i brojem portova za svaki čvor. Kada se mišem klikne na ime čvora, npr. Switch2, otvora se prozor s funkcijama koje su prikazane na slici 19. Klikom mišem na broj porta, prikazati će se svi aktivni portovi određenog *switch*-a, npr. za Switch1 su to eth1, eth2, eth3 i eth4.




Slika 18. GUI *OpenDaylight* kontrolera s naučenim čvorovima

Izvor: Izradio autor




Slika 19. Promjena informacija *switch*-a

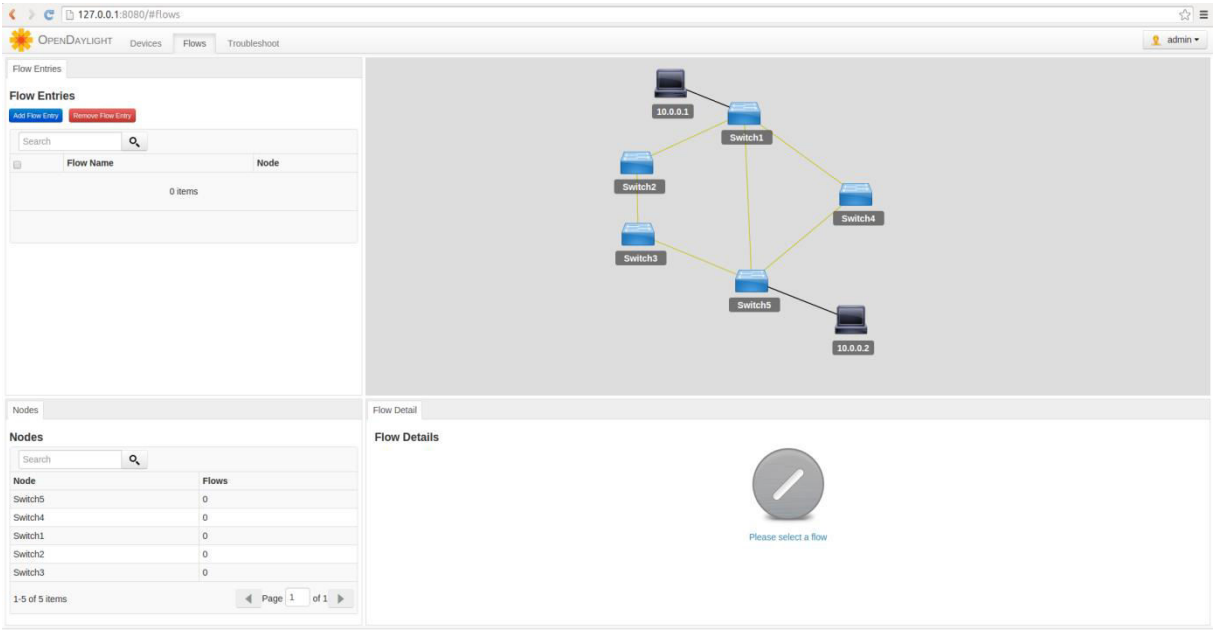
Izvor: Izradio autor

U trenutnom izborniku je moguće još konfigurirati statičke rute i *Gateway*, što je prikazano na slici 20.


Slika 20. Konfiguracija statičke rute i *Gateway*-a

Izvor: Izradio autor

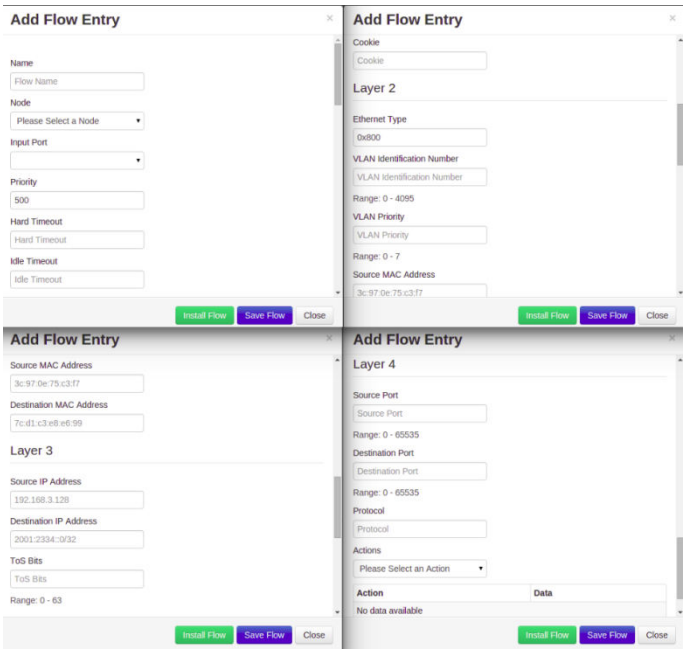
Sljedeći izbornik je *Flows*. Kao što je vidljivo na slici 21., lijevo gore se nalaze tzv. *Flow Entries* koji su dodani te se dolje vidi broj *Flow*-a za svaki *switch* te detalji *Flow*-a, što će biti prikazano u slijedećem poglavlju.




Slika 21. Izgled *Flows* izbornika

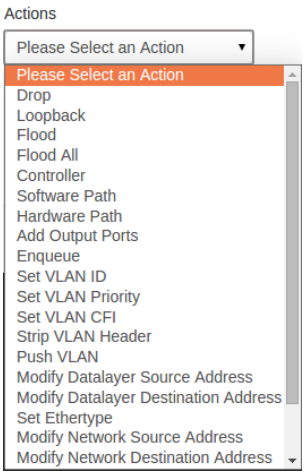
Izvor: Izradio autor

Klikom mišem na plavi gumb *Add Flow Entry* otvara se prozor s funkcijama kreiranja *flow*-a za određeni *switch* u bilo kojem trenutku, koji su prikazani na slici 22. te odabirom radnje određenog *flow*-a na slici 23.




Slika 22. Funkcije pri instalaciji *flow*-a

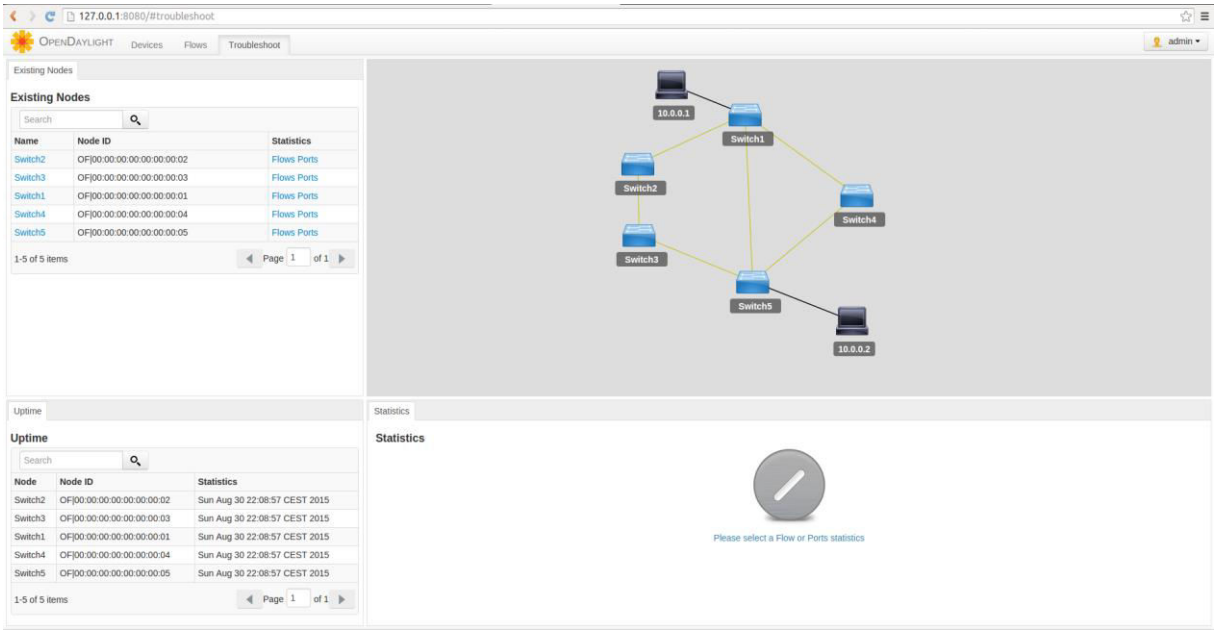
Izvor: Izradio autor




Slika 23. Odabir određene radnje za pojedini *flow*

Izvor: Izradio autor

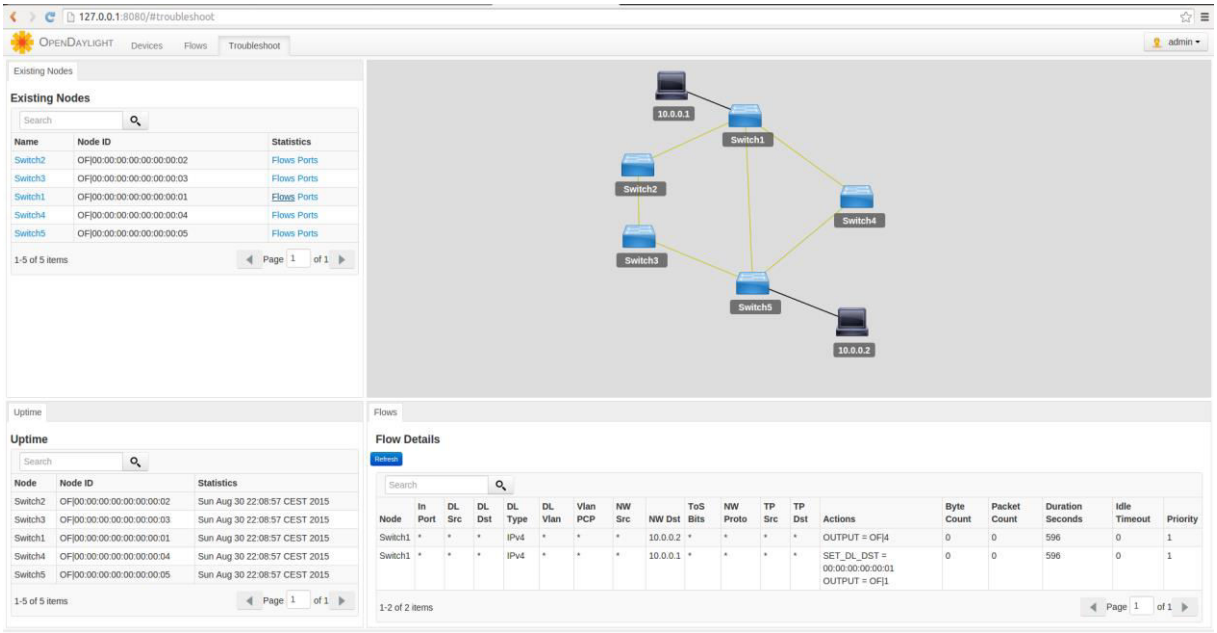
Zadnji izbornik je *Troubleshoot* (slika 24.) koji služi za pregled statistika *flow*-a i porta pojedinog čvora.




Slika 24. Izgled *Troubleshoot* izbornika

Izvor: Izradio autor

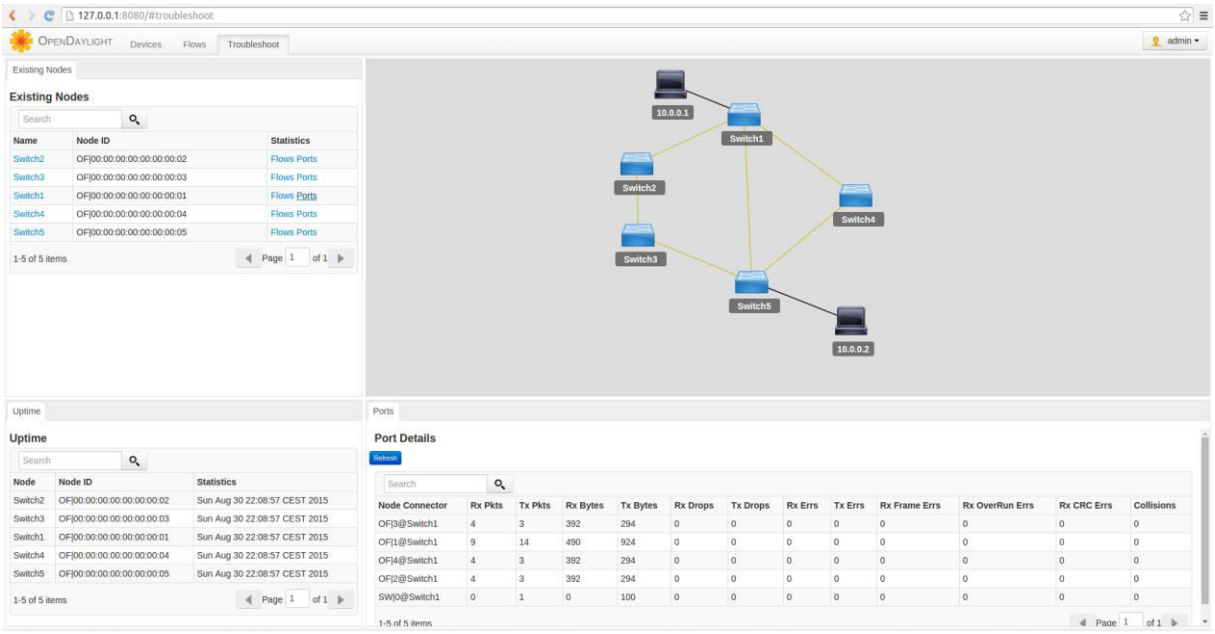
Odabirom statistike *flow*-a za određeni *switch*, prikazati će se detalji *flow*-a što je prikazano na slici 25 te odabirom statistike porta prikazati će se detalji za svaki port odabranog *switch*-a, što je prikazano na slici 26.




Slika 25. Detalji *flow*-a određenog *switch*-a

Izvor: Izradio autor

Neki od detalja koji se mogu ovdje očitati su ulazni port (In Port), *Data Link Layer Source* i *Destination* (DL Src i DL Dst), *Data Link Layer Type* (DL Type), *Network Layer Source* i *Destination* (NW Src i NW Dst), *Transport Protocol Source* i *Destination* (TP Src i TP Dst), radnju koju flow obavlja, brojač *byte*-ova i paketa, prioritet itd.



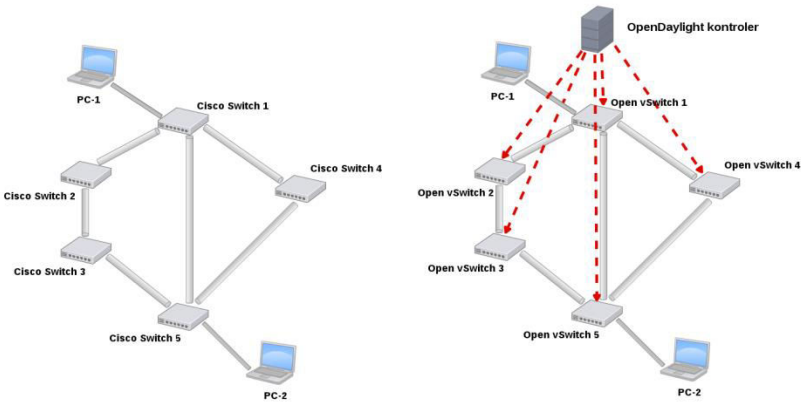

Slika 26. Detalji porta određenog *switch*-a

Izvor: Izradio autor

Ovdje se mogu npr. za odabrani port očitati broj primljenih i poslanih paketa i *byte*-ova, broj odbačenih paketa itd.

## 7.2. Izrada konvencionalne mrežne topologije

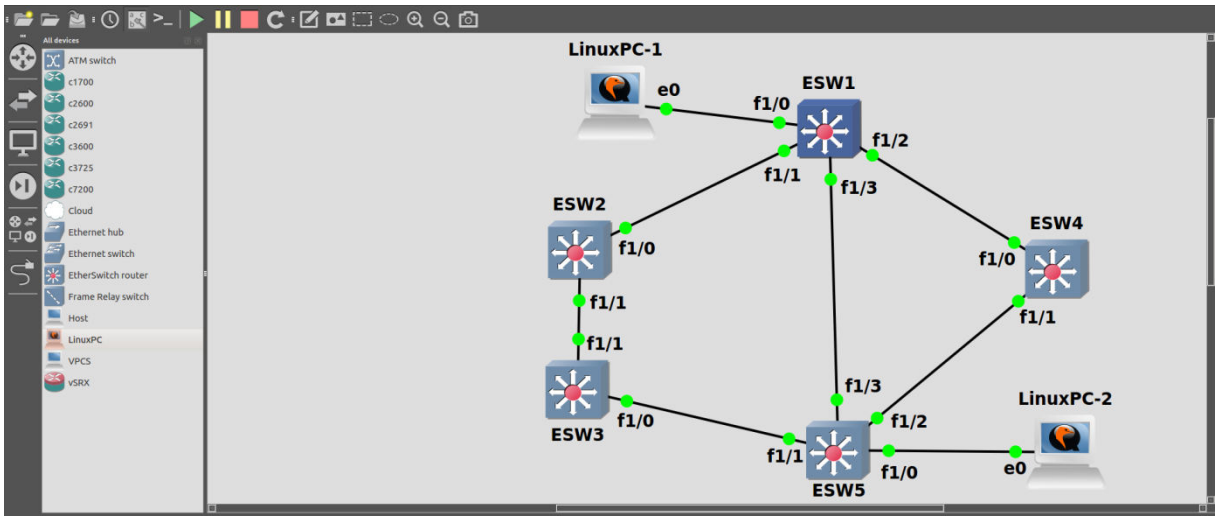
Mrežna topologija korištena za usporedbu konfiguracije konvencionalne i SDN mreže se sastoji od 5 *switch*-eva i 2 Linux PC-a, povezanih na način prikazan na slici 27.




Slika 27. Mrežna topologija konvencionalne i SDN mreže

Izvor: Izradio autor

Korištenjem GNS3 simulatora, kreirana je mreža kao na slici 28. Prije nego bilo koji promet od PC-1 do PC-2 može teći, potrebno je konfigurirati sve *switch*-eve, kako bi promet od PC-1 do PC-2 tekao najkraćim putem, a što se omogućava korištenjem FIB-a (*Forwarding Information Base*) na svakom *switch-u*.




Slika 28. Mrežna topologija konvencionalne mreže u GNS3 simulatoru

Izvor: Izradio autor

Konfiguracija *switch*-a se pokreće otvaranjem *Console* u kojoj se konfigurira sami *switch*. Slike 29., 30., 31., 32. i 33. prikazuju podešavanje svih *switch*-eva s odgovarajućim portima koji su prikazani na slici 28.

```
ESW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
ESW1(config)#interface FastEthernet1/0
ESW1(config-if)# switchport mode access
ESW1(config-if)#no ip address
ESW1(config-if)#no shutdown
ESW1(config-if)# duplex full
ESW1(config-if)# speed 100
ESW1(config-if)#!
ESW1(config-if)#interface FastEthernet1/1
ESW1(config-if)# switchport mode trunk
ESW1(config-if)#no ip address
ESW1(config-if)#no shutdown
ESW1(config-if)# duplex full
ESW1(config-if)# speed 100
ESW1(config-if)#!
ESW1(config-if)#interface FastEthernet1/2
ESW1(config-if)# switchport mode trunk
ESW1(config-if)#no ip address
ESW1(config-if)#no shutdown
ESW1(config-if)# duplex full
ESW1(config-if)# speed 100
ESW1(config-if)#!
ESW1(config-if)#interface FastEthernet1/3
ESW1(config-if)# switchport mode trunk
ESW1(config-if)#no ip address
ESW1(config-if)#no shutdown
ESW1(config-if)#
*Mar  1 00:01:26.395: %DTP-5-TRUNKPORTON: Port Fa1/1 has become dot1q trunk duplex full
ESW1(config-if)# speed 100
ESW1(config-if)#!
*Mar  1 00:01:26.983: %LINK-3-UPDOWN: Interface FastEthernet1/0, changed state to up
*Mar  1 00:01:27.567: %LINK-3-UPDOWN: Interface FastEthernet1/1, changed state to up
*Mar  1 00:01:27.571: %DTP-5-TRUNKPORTON: Port Fa1/2-3 has become dot1q trunk
*Mar  1 00:01:27.983: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/0, changed state to up
*Mar  1 00:01:28.483: %LINK-3-UPDOWN: Interface FastEthernet1/2, changed state to up
ESW1(config-if)#!
*Mar  1 00:01:28.567: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/1, changed state to up
*Mar  1 00:01:29.067: %LINK-3-UPDOWN: Interface FastEthernet1/3, changed state to up
ESW1(config-if)#!
*Mar  1 00:01:29.483: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/2, changed state to up
*Mar  1 00:01:30.067: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/3, changed state to up
ESW1(config-if)#
```

Slika 29. Konfiguracija ESW1 *switch*-a

```
ESW2#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
ESW2(config)#interface FastEthernet1/0
ESW2(config-if)# switchport mode trunk
ESW2(config-if)#no ip address
ESW2(config-if)#no shutdown
ESW2(config-if)# duplex full
ESW2(config-if)# speed 100
ESW2(config-if)#!
ESW2(config-if)#interface FastEthernet1/1
ESW2(config-if)# switchport mode trunk
ESW2(config-if)#no ip address
ESW2(config-if)#no shutdown
ESW2(config-if)# duplex full
ESW2(config-if)# speed 100
ESW2(config-if)#!
*Mar  1 00:08:58.095: %DTP-5-TRUNKPORTON: Port Fa1/0 has become dot1q trunk
*Mar  1 00:08:58.991: %DTP-5-TRUNKPORTON: Port Fa1/1 has become dot1q trunk
ESW2(config-if)#!
*Mar  1 00:08:59.043: %LINK-3-UPDOWN: Interface FastEthernet1/0, changed state to up
*Mar  1 00:09:00.043: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/0, changed state to up
ESW2(config-if)#!
*Mar  1 00:09:00.251: %LINK-3-UPDOWN: Interface FastEthernet1/1, changed state to up
*Mar  1 00:09:01.251: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/1, changed state to up
ESW2(config-if)#
```

Slika 30. Konfiguracija ESW2 *switch*-a

```
ESW3#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
ESW3(config)#interface FastEthernet1/0
ESW3(config-if)# switchport mode trunk
ESW3(config-if)#no ip address
ESW3(config-if)#no shutdown
ESW3(config-if)# duplex full
ESW3(config-if)# speed 100
ESW3(config-if)#!
ESW3(config-if)#interface FastEthernet1/1
ESW3(config-if)# switchport mode trunk
ESW3(config-if)#no ip address
ESW3(config-if)#no shutdown
ESW3(config-if)# duplex full
ESW3(config-if)# speed 100
ESW3(config-if)#!
*Mar  1 00:09:17.387: %DTP-5-TRUNKPORTON: Port Fa1/0 has become dot1q trunk
*Mar  1 00:09:18.303: %DTP-5-TRUNKPORTON: Port Fa1/1 has become dot1q trunk
ESW3(config-if)#!
*Mar  1 00:09:18.335: %LINK-3-UPDOWN: Interface FastEthernet1/0, changed state to up
*Mar  1 00:09:19.335: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/0, changed state to up
ESW3(config-if)#!
*Mar  1 00:09:19.543: %LINK-3-UPDOWN: Interface FastEthernet1/1, changed state to up
*Mar  1 00:09:20.543: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/1, changed state to up
ESW3(config-if)#
```

Slika 31. Konfiguracija ESW3 *switch*-a

```
ESW5#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
ESW5(config)#interface FastEthernet1/0
ESW5(config-if)# switchport mode access
ESW5(config-if)#no ip address
ESW5(config-if)#no shutdown
ESW5(config-if)# duplex full
ESW5(config-if)# speed 100
ESW5(config-if)#!
ESW5(config-if)#interface FastEthernet1/1
ESW5(config-if)# switchport mode trunk
ESW5(config-if)#no ip address
ESW5(config-if)#no shutdown
ESW5(config-if)# duplex full
ESW5(config-if)# speed 100
ESW5(config-if)#!
ESW5(config-if)#interface FastEthernet1/2
ESW5(config-if)# switchport mode trunk
ESW5(config-if)#no ip address
ESW5(config-if)#no shutdown
ESW5(config-if)# duplex full
ESW5(config-if)# speed 100
ESW5(config-if)#!
ESW5(config-if)#interface FastEthernet1/3
ESW5(config-if)# switchport mode trunk
ESW5(config-if)#no ip address
ESW5(config-if)#no shutdown
ESW5(config-if)#
*Mar  1 00:04:41.403: %DTP-5-TRUNKPORTON: Port Fa1/1 has become dot1q trunk duplex full
ESW5(config-if)# speed 100
ESW5(config-if)#!
*Mar  1 00:04:41.991: %LINK-3-UPDOWN: Interface FastEthernet1/0, changed state to up
*Mar  1 00:04:42.575: %DTP-5-TRUNKPORTON: Port Fa1/2-3 has become dot1q trunk
*Mar  1 00:04:42.575: %LINK-3-UPDOWN: Interface FastEthernet1/1, changed state to up
*Mar  1 00:04:42.991: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/0, changed state to up
*Mar  1 00:04:43.491: %LINK-3-UPDOWN: Interface FastEthernet1/2, changed state to up
ESW5(config-if)#!
*Mar  1 00:04:43.575: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/1, changed state to up
*Mar  1 00:04:44.075: %LINK-3-UPDOWN: Interface FastEthernet1/3, changed state to up
ESW5(config-if)#!
*Mar  1 00:04:44.491: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/2, changed state to up
*Mar  1 00:04:45.075: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/3, changed state to up
ESW5(config-if)#
```

Slika 32. Konfiguracija ESW5 *switch*-a

```
ESW4#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
ESW4(config)#interface FastEthernet1/0
ESW4(config-if)# switchport mode trunk
ESW4(config-if)#no ip address
ESW4(config-if)#no shutdown
ESW4(config-if)# duplex full
ESW4(config-if)# speed 100
ESW4(config-if)#!
ESW4(config-if)#interface FastEthernet1/1
ESW4(config-if)# switchport mode trunk
ESW4(config-if)#no ip address
ESW4(config-if)#no shutdown
ESW4(config-if)# duplex full
ESW4(config-if)# speed 100
ESW4(config-if)#!
*Mar  1 00:07:13.979: %DTP-5-TRUNKPORTON: Port Fa1/0 has become dot1q trunk
*Mar  1 00:07:14.879: %DTP-5-TRUNKPORTON: Port Fa1/1 has become dot1q trunk
ESW4(config-if)#!
*Mar  1 00:07:14.931: %LINK-3-UPDOWN: Interface FastEthernet1/0, changed state to up
*Mar  1 00:07:15.931: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/0, changed state to up
ESW4(config-if)#!
*Mar  1 00:07:16.135: %LINK-3-UPDOWN: Interface FastEthernet1/1, changed state to up
*Mar  1 00:07:17.135: %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet1/1, changed state to up
ESW4(config-if)#
```

Slika 33. Konfiguracija ESW4 *switch*-a

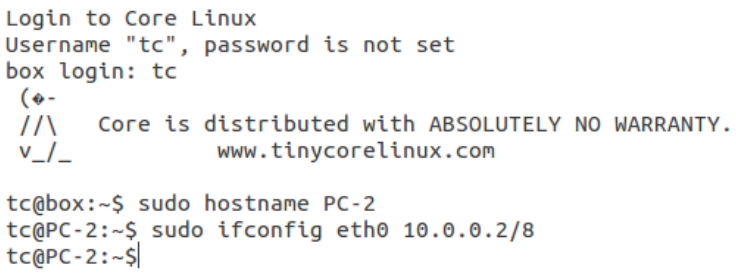
Sa slika se može vidjeti da su *switch*-evi konfigurirani na sljedeći način:

- *switchport mode access* na portima f1/0 ESW1 i ESW5 *switch*-a, pošto su direktno spojeni s PC-1 i PC-2. Svi ostali porti ESW1 i ESW5 te svi porti ostalih ESW2, ESW3 i ESW4 *switch*-eva imaju postavljen *switchport mode trunk*. *Access port* može imati samo jedan VLAN konfiguriran na sučelju i može proslijeđivati promet samo za jedan VLAN. *Trunk port* može imati jedan ili više VLAN-a konfiguriranih na sučelju i može proslijeđivati promet za sve VLAN-ove istovremeno.
- *no ip address*
- *no shutdown* što znači da je port podignut
- *duplex full*
- *speed* 100 što označava brzinu od 100 Mbps

Nakon konfiguracije svih *switch*-eva i njihovih portova, potrebno je postaviti porte na PC-1 i PC-2 te dodati im IP adrese, prikazano na slikama 34. i 35.

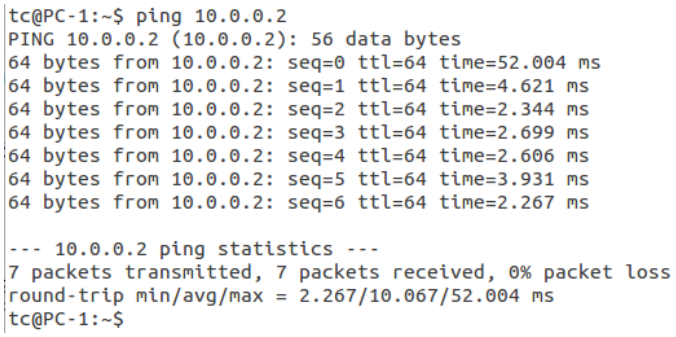
```
Login to Core Linux
Username "tc", password is not set
box login: tc
 (◆-
 //\   Core is distributed with ABSOLUTELY NO WARRANTY.
 v_/_           www.tinycorelinux.com

tc@box:~$ sudo hostname PC-1
tc@PC-1:~$ sudo ifconfig eth0 10.0.0.1/8
tc@PC-1:~$
```

Slika 34. Postavljanje porta i dodjela IP adrese za PC-1




Slika 35. Postavljanje porta i dodjela IP adrese za PC-2

Nakon što su dodijeljeni porti, IP adrese i konfigurirani svi *switch*-evi, mrežna topologija mora biti naučena od svih *switch*-eva, uostalom i za odabir najkraćeg puta između PC-1 i PC-2. To se može učiniti jednostavnim *flow*-om kao što je *ping*[12], od PC-1 do PC-2 (slika 34.). Kada PC-1 želi slati promet prema PC-2, pošalje *ARP-REQUEST* (dest. mac ff:ff:ff:ff:ff:ff) da bi saznao MAC adresu od PC-2. *Switch*, budući da u samom početku u svojoj MAC tabeli nema to saznanje, taj zahtjev *flood*-a na sve porte, osim na *source* port (tamo gdje je spojen PC-1) i taj *ARP-REQUEST* onda "putuje" do PC-2. PC-2 odgovori sa *ARP-REPLY* porukom u kojoj kaže da je njegova IP adresa na toj i toj MAC adresi. Svi *switch*-evi sada redom zapamte na koji port im je došla ta poruka i onda tu MAC adresu u *mac-address* tabeli "smjeste" na taj određeni port, tako da ubuduće znaju na koji port "izbaciti" *frame* prema toj MAC adresi. Ako postoje dva ili više puteva od *source*-a do *destination*-a, da se ne bi stvorila petlja ili "*loop*", koristi se tzv. *Spanning Tree Protocol*[13] (STP), koji prema određenom algoritmu onda blokira određene porte na *switch*-evima, tako da je u svakom trenutku moguć samo jedan put od PC-1 do PC-2.




Slika 36. Rezultati *ping*-a od PC-1 do PC-2

[12] Ping: Ping je administrativni alat koji služi za provjeru dostupnosti host-ova na računalnim mrežama temeljenim na IP protokolu.
[13] Spanning Tree Protocol: Spanning Tree Protocol je protokol za upravljanje linkovima (implementiran na 2. sloju OSI modela), a glavna mu je zadaća osiguravanje redundantnih linkova i sprječavanja pojave petlji u topologiji.

Iz slike se može zaključiti da je konfiguracija bila uspješna. Treba još utvrditi da li je izabran najbolji put od PC-1 do PC-2 koji bi trebao voditi preko ESW1 i porta f1/0 i f1/3 te ESW5 i porta f1/3 i f1/0. To se može saznati uvidom u MAC *address* tabeli na *switch*-evima. Prije toga treba provjeriti MAC adrese pojedinih PC-a (slike 37. i 38.).

```
tc@PC-1:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:00:AB:49:91:00
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
```

Slika 37. MAC adresa PC-1

```
tc@PC-2:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:00:AB:48:F8:00
          inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
```

Slika 38. MAC adresa PC-2

Iz slika se može pročitati da je MAC adresa PC-1 00:00:AB:49:91:00, a za PC-2 00:00:AB:48:F8:00. Sada je potrebno provjeriti da li navedene MAC adrese odgovaraju pojedinim portima *switch*-eva.

```
ESW1#show mac-address-table
Destination Address  Address Type  VLAN  Destination Port
-------------------  ------------  ----  --------------------
c201.2251.0000          Self          1     Vlan1
0000.ab49.9100          Dynamic       1     FastEthernet1/0
0000.ab48.f800          Dynamic       1     FastEthernet1/3
```

Slika 39. MAC *address* tabela za ESW1

```
ESW5#show mac-address-table
Destination Address  Address Type  VLAN  Destination Port
-------------------  ------------  ----  --------------------
c205.2294.0000          Self          1     Vlan1
0000.ab49.9100          Dynamic       1     FastEthernet1/3
0000.ab48.f800          Dynamic       1     FastEthernet1/0
```

Slika 40. MAC *address* tabela za ESW5

Na slikama 39. i 40. se može jasno uočiti da je izabran najbolji put, pošto MAC adresa PC-1 odgovara portu f1/0 na ESW1 i portu f1/3 na ESW5 te MAC adresa PC-2 odgovara portu f1/3 na ESW1 i portu f1/0 na ESW5 te je time *flow* tekao kroz odgovarajuće *switch*-eve i najkraćim putem.

Iako se mrežna topologija sastoji od malog broja mrežnih uređaja, nije previše složena te je kreiran vrlo jednostavan *flow*, može se zaključiti da konfiguracija većih mreža zahtijeva dosta više postupaka i vremena. Ukoliko se mreža sastoji od tisuća mrežnih uređaja i *host*-ova, što je česti slučaj u današnjim mrežama, svaki *switch* se za odgovarajući promet, *flow* i svaku promjenu *flow*-a mora posebno konfigurirati, što naravno povećava broj postupaka i vrijeme potrebno za to. Ako se još dodaju VLAN-ovi i QoS, posao administratora se dodatno povećava, jer se potrebna konfiguracija mora napraviti na svakom *switch-u*, a povećava se i mogućnost pogreške. Isto tako porastom broja mrežnih uređaja raste i FIB na svakom uređaju, a time i potreba za povećanjem njihove radne memorije, što dodatno povećava njihovu nabavnu cijenu i trošak operatora.

U SDN rješenju sve navedene postupke, od konfiguriranja *switch*-eva i učenja mrežne topologije, obavlja SDN kontroler sa jednog centraliziranog mjesta i to u vrlo kratkom roku te u tome leži velika prednost u odnosu na konvencionalne mreže. *Switch* mora biti spojen na SDN kontroler te sve ostalo obavlja sam kontroler. Kako točno SDN kontroler radi prikazano je u nastavku.

## 7.3. Izrada SDN mrežne topologije

Izrada SDN mrežne topologije obavljena je pomoću *Mininet* emulatora koji je pokrenut na *Linux* virtualnoj mašini, dok je *OpenDaylight* kontroler pokrenut na *host Linux* računalu. Topologija se sastoji od 5 *Open vSwitch*-a i 2 *Linux* PC-a povezanih kao kod izrade konvencionalne mreže. Kreiranje mrežne topologije prikazano je na slici 41. Radi jednostavnijeg rada iz *host* računala, pokrenuta je SSH veza između *host* računala i Mininet emulatora, što naravno nije nužno.

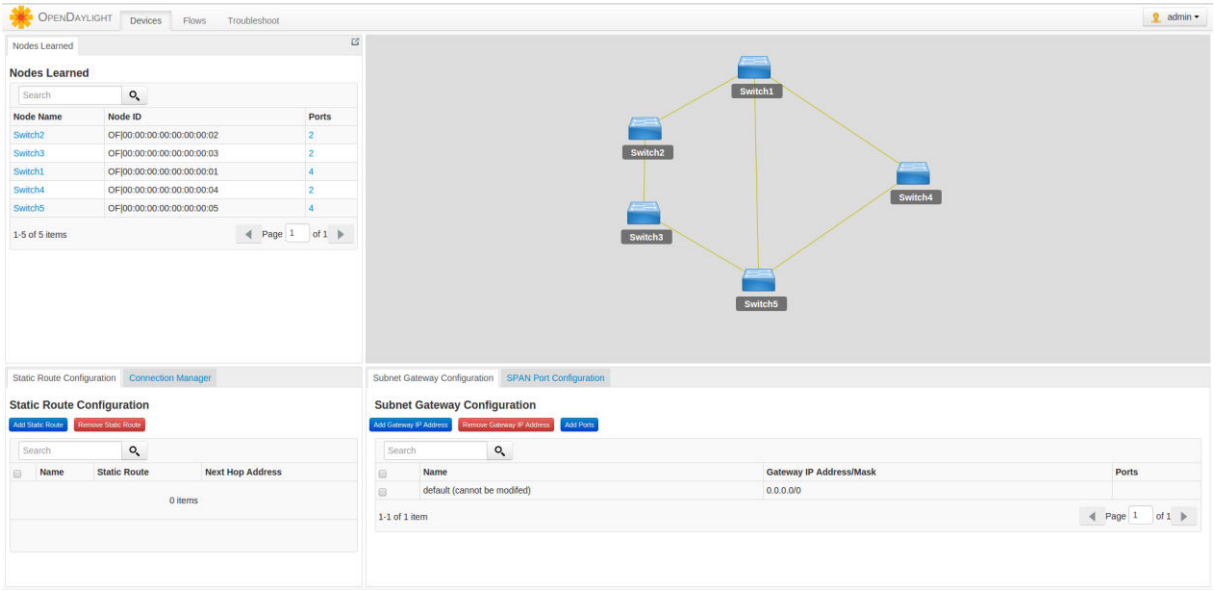
Nakon uspješne SSH veze, potrebno je ući u odgovarajući direktorij *mininet/custom* gdje se nalazi vlastita topologija kreirana za ovaj slučaj, napisana u *Python* programskom jeziku pod nazivom *MojaTopologija*. Naredba za pokretanje kreiranja mreže glasi *sudo mn –mac –controller=remote,ip=192.168.165.1,port=663 –custom MojaTopologija.py –topo=mytopo*, a pojedini dijelovi naredbe znače sljedeće:

- sudo mn: Pokreće komandu s *root* privilegijem
- --mac: Postavlja MAC adrese *host*-ova sličnim IP adresama, što olakšava npr. čitanje praćenog prometa u *Wireshark*-u
- --controller=remote: Javlja *Mininet*-u da se SDN kontroler ne nalazi na lokalnom računalu
- Ip=192.168.165.1: IP adresa SDN kontrolera, u ovom slučaju i IP adresa *host* računala na kojem se kontroler pokreće
- Port=6633: Standardni TCP port za povezivanje *switch*-a na kontroler
- --custom MojaTopologija.py –topo=mytopo: Pokreće vlastitu topologiju napisanu u *Python* jeziku

```
anthony@anthony:~$ ssh -X mininet@192.168.165.130
mininet@192.168.165.130's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Tue Sep  1 13:21:57 2015 from 192.168.165.1
mininet@mininet-vm:~$ cd mininet/custom/
mininet@mininet-vm:~/mininet/custom$ sudo mn --mac --controller=remote,ip=192.168.165.1,
port=6633 --custom MojaTopologija.py --topo=mytopo
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s5) (s1, s2) (s1, s4) (s1, s5) (s2, s3) (s3, s5) (s4, s5)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet>
```

Slika 41. Kreiranje SDN mreže pomoću *Mininet* emulatora

Ukoliko je sve dobro napisano, tj. pravilnim kodom, *Mininet* će kreirati mrežu dodavanjem kontrolera, *host*-ova, *switch*-eva i linkova te će konfigurirati *host*-ove i pokrenuti *switch*-eve.




Slika 42. Prikaz naučenih čvorova SDN kontrolera

Izvor: Izradio autor

Slika 42. prikazuje vidljivost svih *switch*-eva i njihovu povezanost u *OpenDaylight* kontroleru, što znači da je uspješno naučio topologiju te mogućnosti svih *switch*-eva. Iako se ovdje radi o izradi virtualne mreže, što se tiče *switch*-eva, korišteni kontroler se koristi i u stvarnim fizičkim mrežama. Vidljivo je koliko olakšava posao u odnosu na konvencionalnu mrežu, gdje se ručno mora konfigurirati svaki *switch*, što u SDN-u obavlja kontroler na vrlo brz način, što je omogućeno odvajanjem kontrolne cjeline u obliku *software*-a kojim upravlja kontroler, od podatkovne cjeline koja se dalje nalazi u samom *switch*-u. Otkrivanje topologije zajedno s *switch*-evima i linkovima zahtijeva određene korake koji su objašnjeni u nastavku.

### 7.3.1. Otkrivanje topologije

Prvi korak *OpenFlow* mreže je otkriće *OpenFlow switch*-eva putem kontrolera. Kontroler treba znati topologiju prije nego što zapravo može tražiti put između dva različita *host*-a u *OpenFlow* mreži i instalirati bilo koji *flow* za promet podatkovne cjeline. *OpenFlow* kontroler uči o *OpenFlow switch*-evima slušajući na TCP portu broj 6633. Nakon što kontroler zna za *switch*, sljedeći korak je otkriti ukupni pogled na mrežu (tj. saznati pojedine detalje *OpenFlow switch*-a te veze između različitih

*switch*-eva). Ovo otkriće je učinjeno u dva koraka: Prvi korak je saznanje o pojedinim *switch*-evima, a drugi korak je saznanje povezanosti tih *switch*-eva.

Pomoću *feature request* i *feature reply* mehanizama je poduzet prvi korak. Kontroler šalje *feature-request* poruku čim je obavljen tzv. TCP *handshake*. Novo povezani *switch* odgovora s *feature-reply* porukom. *Feature-reply* poruka govori kontroleru o sposobnostima *switch*-a, detaljima porta i sposobnostima radnji. U drugom koraku, otkrivanje međusobnih veza *switch*-eva događa se pomoću *Link Layer Discovery Protocol*[14] (LLDP) okvira, koji su poslani na sve povezane (podignute) porte *switch*-eva između različitih *OpenFlow switch*-eva.

Po definiciji *OpenFlow* protokola, *OpenFlow switch* ne razumije LLDP, tako da su svi LLDP okviri zapravo napravljeni od kontrolera i poslani pomoću *packet-out* događaja iz kontrolera, s radnjom da se šalju na sve ili podignute porte. Dolazni LLDP paketi na povezanim portima *peer switch*-a su enkapsulirani u *packet-in* zaglavlja te poslani od *switch*-eva prema kontroleru. Nakon što je kontroler primio sve pakete, imati će pogled na potpunu topologiju (nakon što obrađuje razne LLDP pakete koji dolaze od raznih *switch*-eva).

Detalji o tome preko kojih porta su pojedini *switch*-evi povezani s drugim i s *host*-ovima, mogu e dobiti naredbom *links* u CLI-u *Mininet*-a.

```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s5-eth1 (OK OK)
s1-eth2<->s2-eth1 (OK OK)
s1-eth3<->s4-eth1 (OK OK)
s1-eth4<->s5-eth2 (OK OK)
s2-eth2<->s3-eth1 (OK OK)
s3-eth2<->s5-eth3 (OK OK)
s4-eth2<->s5-eth4 (OK OK)
mininet>
```

Slika 43. Povezanost mrežne topologije

Na temelju gore navedenih koraka, kontroler može imati pogled na topologiju prikazanoj na slici 42. i cijela operacija otkrivanja te topologije navedena je u tablici 2. sa sljedećim koracima:

[14] Link Layer Discovery Protocol : Link Layer Discovery Protocol (LLDP) je vendor-neutral link layer protokol koji je korišten od mrežnih uređaja za oglašavanje svog identiteta, mogućnosti i susjednih uređaja.

Tablica 2. Prikaz postupka otkrivanja topologije od strane kontrolera

| Paket pokrenut od: | Paket određen za: | Tip paketa | Paket payload i detalji |
|---|---|---|---|
| Kontrolera | Sve switch-eve | Feature request | Kontroler šalje feature request prema svim switch-evima |
| Svih switch-eva | Kontroler | Feature reply | Svi switch-evi šalju feature-reply poruku |
| Kontrolera | Switch1 | Packet out | Kontroler pita Switch1 da šalje LLDP pakete na port eth1, eth2, eth3 i eth4 |
| Kontrolera | Switch2 | Packet out | Kontroler pita Switch2 da šalje LLDP pakete na port eth1 i eth2 |
| Kontrolera | Switch3 | Packet out | Kontroler pita Switch3 da šalje LLDP pakete na port eth1 i eth2 |
| Kontrolera | Switch4 | Packet out | Kontroler pita Switch4 da šalje LLDP pakete na port eth1 i eth2 |
| Kontrolera | Switch5 | Packet out | Kontroler pita Switch5 da šalje LLDP pakete na port eth1, eth2, eth3 i eth4 |
| Switch1 | Kontroler | Packet in | Switch1 šalje enkapsulirani LLDP paket kojeg je primio od Switch2 na port-u eth2 |
| Switch1 | Kontroler | Packet in | Switch1 šalje enkapsulirani LLDP paket kojeg je primio od Switch4 na port-u eth3 |
| Switch1 | Kontroler | Packet in | Switch1 šalje enkapsulirani LLDP paket kojeg je primio od Switch5 na port-u eth4 |
| Switch2 | Kontroler | Packet in | Switch2 šalje enkapsulirani LLDP paket kojeg je primio od Switch1 na port-u eth1 |
| Switch2 | Kontroler | Packet in | Switch2 šalje enkapsulirani LLDP paket kojeg je primio od Switch3 na port-u eth2 |
| Switch3 | Kontroler | Packet in | Switch3 šalje enkapsulirani LLDP paket kojeg je primio od Switch2 na port-u eth1 |
| Switch3 | Kontroler | Packet in | Switch3 šalje enkapsulirani LLDP paket kojeg je primio od Switch5 na port-u eth2 |
| Switch4 | Kontroler | Packet in | Switch4 šalje enkapsulirani LLDP paket kojeg je primio od Switch1 na port-u eth1 |
| Switch4 | Kontroler | Packet in | Switch4 šalje enkapsulirani LLDP paket kojeg je primio od Switch5 na port-u eth2 |
| Switch5 | Kontroler | Packet in | Switch5 šalje enkapsulirani LLDP paket kojeg je primio od Switch1 na port-u eth2 |
| Switch5 | Kontroler | Packet in | Switch5 šalje enkapsulirani LLDP paket kojeg je primio od Switch3 na port-u eth3 |
| Switch5 | Kontroler | Packet in | Switch5 šalje enkapsulirani LLDP paket kojeg je primio od Switch4 na port-u eth4 |

Izvor: Izradio autor

Nakon što je topologija uspješno otkrivena, potrebno je otkriti *host*-ove koji su spojeni na *switch*-evima kako bi kontroler mogao izračunati najbolji put. Postupak otkrivanja *host*-ova i najboljeg puta opisano je u sljedećem potpoglavlju.

### 7.3.2. Otkrivanje host-ova i najboljeg puta

Sada kada su otkriveni svi *switch*-evi i njihova povezanost, tj. nakon što je kontroler naučio topologiju, potrebno je da kontroler otkrije krajnje *host*-ove kako bi mogao odrediti najbolji put. Ovdje je također potrebno kreirati određeni *flow* od PC-1 (h1) do PC-2 (h2) koji sadrži određeni zahtjev kao što je *ARP-REQUEST*. To se može također učiniti pomoću *ping*-a od PC-1 do PC-2 (slika 44.). Koraci obrade *ARP-REQUEST*-a se razlikuju u odnosu na konvencionalnu mrežu zbog prisutnosti SDN kontrolera.

Kada se *ping* naredba pokreće iz PC-1 prema PC-2, *ARP-REQUEST* će biti generiran od PC-1, kojeg će poslati na Switch1. Kada Switch1 dobije *ARP-REQUEST*, poslati će paket (koji će imati enkapsulirani *ARP-REQUEST* u sebi) prema kontroleru. Dobivanjem paketa, kontroler će obaviti dekapsulaciju paketa i vidjeti *ARP-REQUEST*. Kontroler će poslati *packet-out* (s izvornim *ARP-REQUEST*-om) prema svim rubnim *switch*-evima s radnjom da se *ARP-REQUEST* šalje prema svi rubnim portima. U topologiji, jedini rubni *switch* je Switch5 (Switch2, Switch3 i Switch4 su tranzitni *switch*-evi). Kako kontroler zna da li se radi o rubnom ili tranzitnom *switch*-u? Kada se dogodi razmjena LLDP-a, LLDP paketi se šalju prema svim portima koji se nalaze u podignutom stanju. Ako *switch* ne dobije *reply* LLDP-a poslan na "neki" od portova s podignutim stanjem, poznato je da su ti "neki" podignuti porti rubni porti koji nisu povezani sa *switch*-em, nego s drugim uređajem (u ovom slučaju PC).

Kada Switch5 dobije *paket-out* i prati radnju proslijeđivanja izvornog *ARP-REQUEST*-a prema svim rubnim portima, dobit će *reply* od PC-2. Switch5 prosljeđuje *ARP-REPLY* kontroleru, zapravo šalje *ARP-REPLY* u obliku *packet-in* prema kontroleru, s MAC adresom od PC-2.

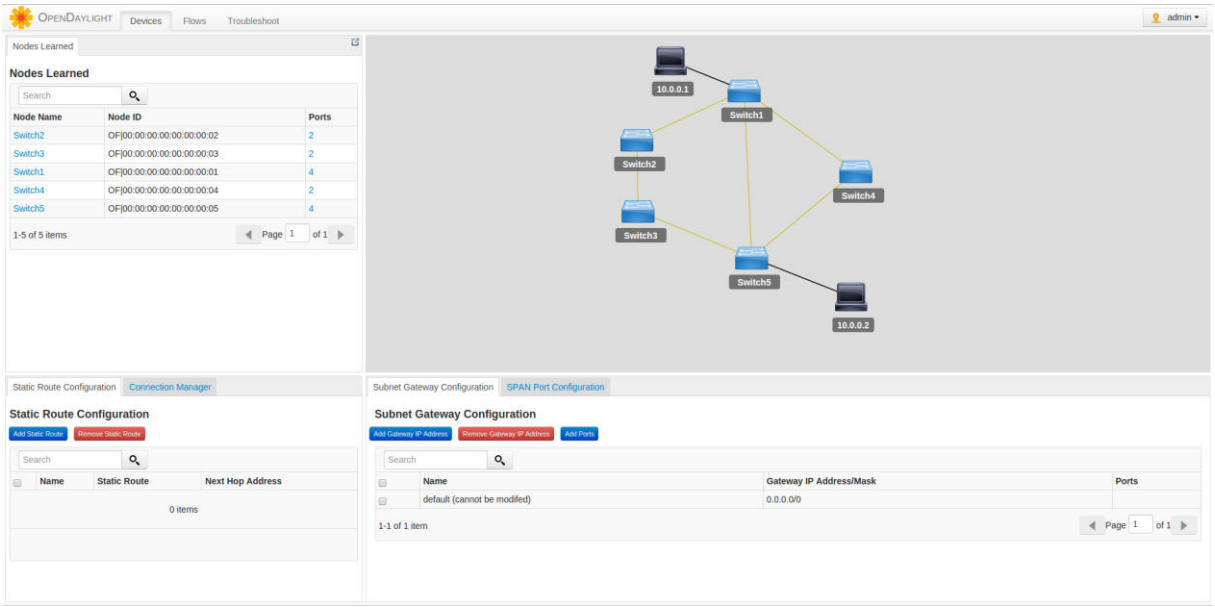
Kontroler obavještava Switch1 o MAC adresi PC-2, tako da PC-1 može popuniti svoju ARP tablicu. Kontroler šalje *packet-out* prema Switch1 koji sadrži *ARP-REPLY* s radnjom da se šalje prema PC-1. Sada kada PC-1 ima ARP adresu od PC-2, poslati će ICMP *request* paket prema PC-2. Kada ICMP paket stiže do Switch1, enkapsulirati će ga u *packet-in* i poslati kontroleru.

Dok ICMP *request* paket dolazi iz PC-1, u isto vrijeme, kontroler pokreće instaliranje *flow*-a (sada zna MAC adresu pomoću ARP-a). *Flow-modification* paketi su poslani od kontrolera prema svim *switch*-evima, kako bi instalirali *flow* na svakom *switch*-u. *Packet-in* koji je poslan kontroleru od Switch1, enkapsuliranjem ICMP *request*-a, biti će enkapsuliran u *packet-out* i poslan od kontrolera prema Switch5.

*Reply* ICMP paketa biti će poslan od Switch5 prema Switch1 na temelju *flow*-a instaliranog u prijašnjim koracima.

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=27.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.692 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.101 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.085 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.095 ms

--- 10.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6002ms
rtt min/avg/max/mdev = 0.084/4.159/27.965/9.720 ms
```

Slika 44. Rezultati *ping*-a od PC-1 do PC-2

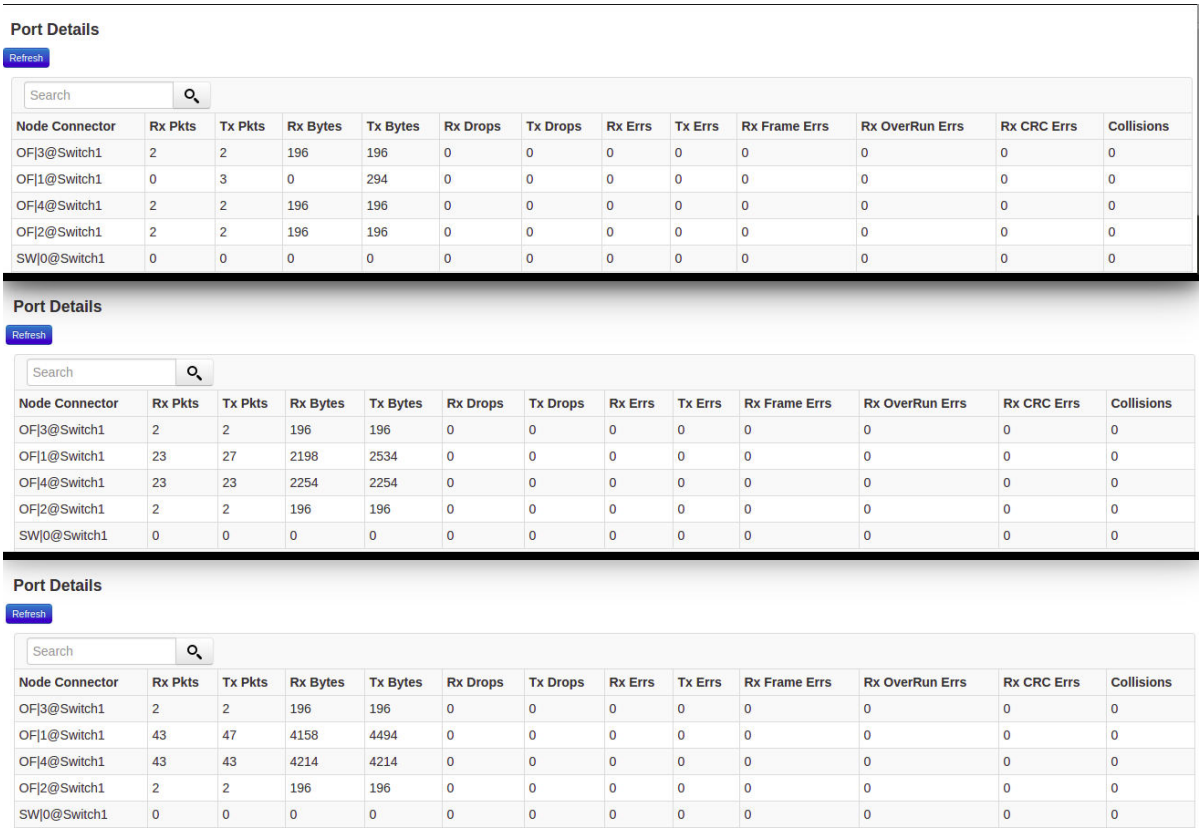



Slika 45. Naučena mrežna topologija zajedno s *host*-ovima

Izvor: Izradio autor

Slika 44. prikazuje uspješnost *ping*-a između PC-1 (h1) i PC-2 (h2) te se na slici 45. može uočiti da uspješan *ping* vodi do toga da je kontroler otkrio *host*-ove h1 i h2 koje samo "vidi".

Prema vidljivoj topologiji i poznatim vezama između *switch*-eva i *host*-ova (slika 45.), najbolji put bi trebao voditi preko Switch1 (porti eth1 i eth4) i Switch5 (porti eth2 i eth1). To se može provjeriti u *OpenDaylight* kontroleru odabirom *Troubleshoot* u izborniku te se za određeni *switch* pogledaju detalji porta (kao što je objašnjeno u poglavlju 7.1., str. 43).



**Port Details**

Refresh

Search

| Node Connector | Rx Pkts | Tx Pkts | Rx Bytes | Tx Bytes | Rx Drops | Tx Drops | Rx Errs | Tx Errs | Rx Frame Errs | Rx OverRun Errs | Rx CRC Errs | Collisions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OF\|3@Switch1 | 2 | 2 | 196 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|1@Switch1 | 0 | 3 | 0 | 294 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|4@Switch1 | 2 | 2 | 196 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|2@Switch1 | 2 | 2 | 196 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SW\|0@Switch1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Port Details**

Refresh

Search

| Node Connector | Rx Pkts | Tx Pkts | Rx Bytes | Tx Bytes | Rx Drops | Tx Drops | Rx Errs | Tx Errs | Rx Frame Errs | Rx OverRun Errs | Rx CRC Errs | Collisions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OF\|3@Switch1 | 2 | 2 | 196 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|1@Switch1 | 23 | 27 | 2198 | 2534 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|4@Switch1 | 23 | 23 | 2254 | 2254 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|2@Switch1 | 2 | 2 | 196 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SW\|0@Switch1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Port Details**

Refresh

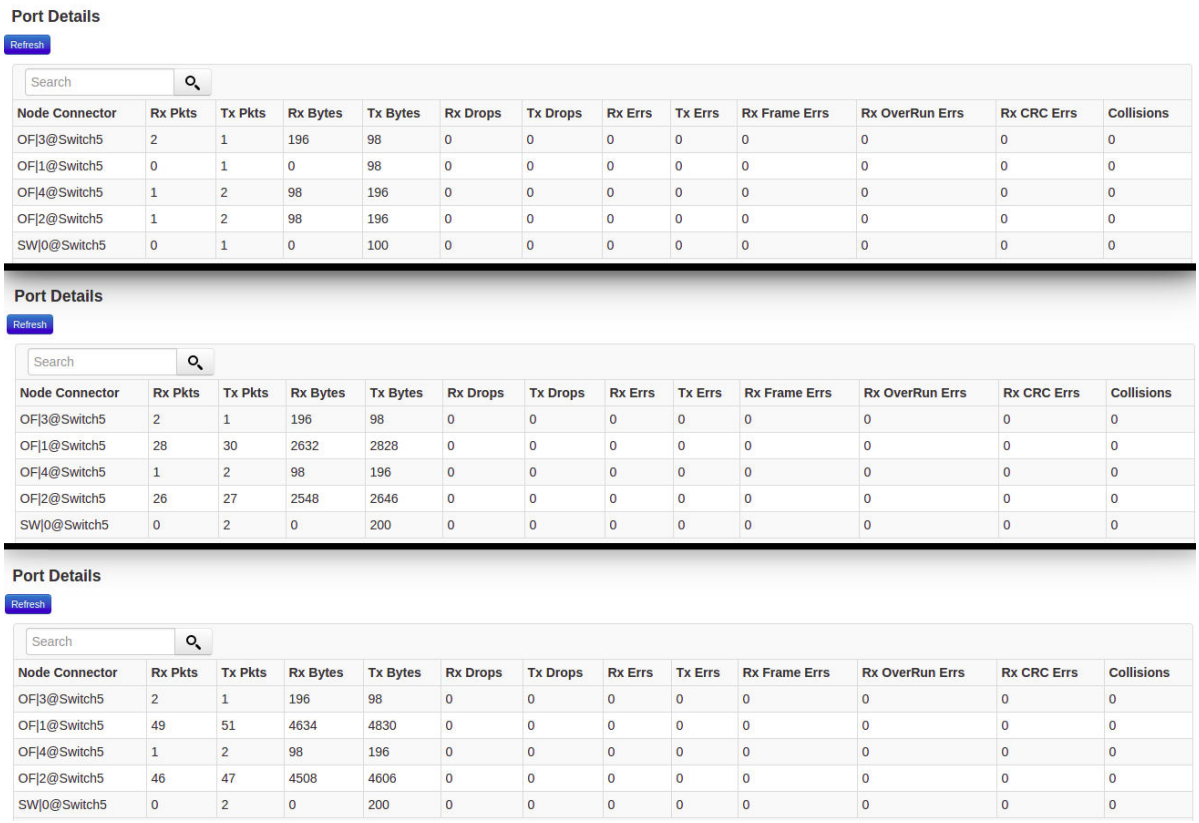
Search

| Node Connector | Rx Pkts | Tx Pkts | Rx Bytes | Tx Bytes | Rx Drops | Tx Drops | Rx Errs | Tx Errs | Rx Frame Errs | Rx OverRun Errs | Rx CRC Errs | Collisions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OF\|3@Switch1 | 2 | 2 | 196 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|1@Switch1 | 43 | 47 | 4158 | 4494 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|4@Switch1 | 43 | 43 | 4214 | 4214 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|2@Switch1 | 2 | 2 | 196 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SW\|0@Switch1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Slika 46. Detalji porta za Switch1

Izvor: Izradio autor

Na slikama 46. i 47. su prikazani detalji porta za Switch1 i za Switch5. Stupac Rx Pkts (*Received Packets*) prikazuje broj primljenih paketa, stupac Tx Pkts (*Transmitted Packets*) prikazuje broj prenesenih paketa a stupci Rx Bytes (*Received Bytes*) i Tx Bytes (*Transmitted Bytes*) prikazuju broj primljenih odnosno broj prenesenih *byte*-ova na pojedinim portima.



Port Details

Refresh

Search

| Node Connector | Rx Pkts | Tx Pkts | Rx Bytes | Tx Bytes | Rx Drops | Tx Drops | Rx Errs | Tx Errs | Rx Frame Errs | Rx OverRun Errs | Rx CRC Errs | Collisions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OF\|3@Switch5 | 2 | 1 | 196 | 98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|1@Switch5 | 0 | 1 | 0 | 98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|4@Switch5 | 1 | 2 | 98 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|2@Switch5 | 1 | 2 | 98 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SW\|0@Switch5 | 0 | 1 | 0 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Port Details

Refresh

Search

| Node Connector | Rx Pkts | Tx Pkts | Rx Bytes | Tx Bytes | Rx Drops | Tx Drops | Rx Errs | Tx Errs | Rx Frame Errs | Rx OverRun Errs | Rx CRC Errs | Collisions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OF\|3@Switch5 | 2 | 1 | 196 | 98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|1@Switch5 | 28 | 30 | 2632 | 2828 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|4@Switch5 | 1 | 2 | 98 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|2@Switch5 | 26 | 27 | 2548 | 2646 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SW\|0@Switch5 | 0 | 2 | 0 | 200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Port Details

Refresh

Search

| Node Connector | Rx Pkts | Tx Pkts | Rx Bytes | Tx Bytes | Rx Drops | Tx Drops | Rx Errs | Tx Errs | Rx Frame Errs | Rx OverRun Errs | Rx CRC Errs | Collisions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OF\|3@Switch5 | 2 | 1 | 196 | 98 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|1@Switch5 | 49 | 51 | 4634 | 4830 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|4@Switch5 | 1 | 2 | 98 | 196 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| OF\|2@Switch5 | 46 | 47 | 4508 | 4606 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SW\|0@Switch5 | 0 | 2 | 0 | 200 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Slika 47. Detalji porta za Switch5

Izvor: Izradio autor

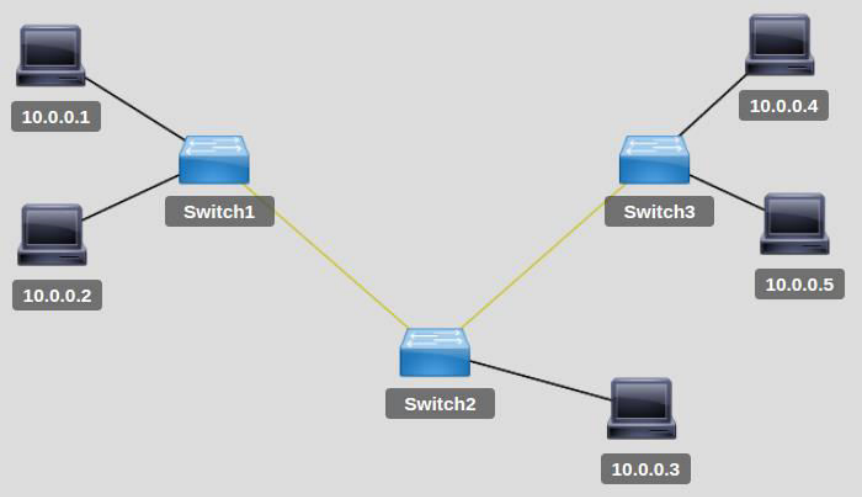
Vidljivo je da se brojevi, kako je *ping* tekao, povećavaju samo na portima eth1 i eth4 kod Switch1 te na portima eth2 i eth1 kod Switch5, što znači da je promet tekao samo preko tih porta i time je kontroler odabrao najbolji, tj. najkraći put.

Koliko brzo traje preusmjeravanje prometa kod konvencionalne mreže u odnosu na SDN mrežu, ukoliko najkraći put više nije dostupan, npr. zbog prekinutog linka, biti će prikazano u nastavku ovog rada.

### 7.3.3. Dodavanje Flow Entry-a

U ovom će se poglavlju prikazati na temelju dva jednostavna primjera, kako se pomoću kontrolera mogu zabraniti određeni protoci za odgovarajuću vrstu prometa. Prvi primjer će prikazati blokiranje ICMP protokola od jednog određenog *host*-a do drugog, dok za ostale *host*-ove neće biti blokiranja. U drugom primjeru će se prikazati blokiranje TCP protokola prema portu 80 jednog *web* poslužitelja.

Slika 48. prikazuje topologiju za prvi primjer. Blokirati će se protok od *host*-a h1 s IP adresom 10.0.0.1 prema *host*-u h4 s IP adresom 10.0.0.4. Instalacija *Flow Entry*-a će se izvršiti na Switch2.



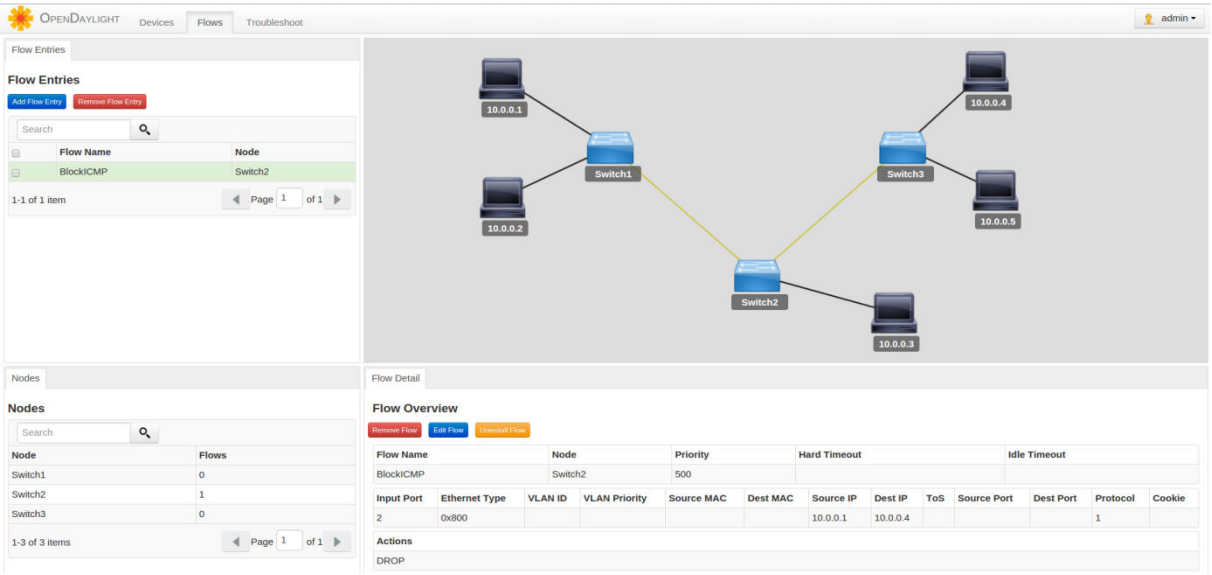

Slika 48. Topologija korištena za *Flow Entry* blokiranja ICMP protokola

Izvor: Izradio autor

Parametri koji su izabrani pri instalaciji *Flow Entry*-a koji su prikazani u poglavlju 7.1. su sljedeći:

- *Name*: BlockICMP
- *Node*: Switch2
- *Input* Port: s2.eth2
- *Priority*: 500
- *Ethernet Type*: 0x800
- *Source IP Address*: 10.0.0.1
- *Destination IP Address*: 10.0.0.4
- *Protocol*: 1 (*Protocol Number* za ICMP)
- *Action*: *Drop*

*Flow Entry* je sada vidljiv u *OpenDaylight* kontroleru, što je prikazano na slici 49.




Slika 49. *Flow Entry* za blokiranje ICMP protokola

Izvor: Izradio autor

Nakon završene instalacije, *ping* ne bi trebao biti moguć samo za relaciju od h1 <-> h4, dok bi npr. za relaciju h1 <-> h5 trebao biti moguć, što vrijedi i za sve ostale relacije. Provjeriti će se prvo relacija h1 <-> h4.

```
mininet> h1 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.

--- 10.0.0.4 ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 8999ms
```

Slika 50. Neuspješan *ping* od h1 do h4

Prema očekivanju *ping* za relaciju h1 <-> h4 je bio neuspješan. Treba još provjeriti da li će *ping* biti uspješan za relaciju h1 <-> h5 te za dodatnu provjeru i za relaciju h2 <-> h4.

```
mininet> h1 ping h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=0.567 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.121 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.096 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=0.114 ms
64 bytes from 10.0.0.5: icmp_seq=5 ttl=64 time=0.127 ms
64 bytes from 10.0.0.5: icmp_seq=6 ttl=64 time=0.106 ms

--- 10.0.0.5 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4999ms
rtt min/avg/max/mdev = 0.096/0.188/0.567/0.170 ms
```

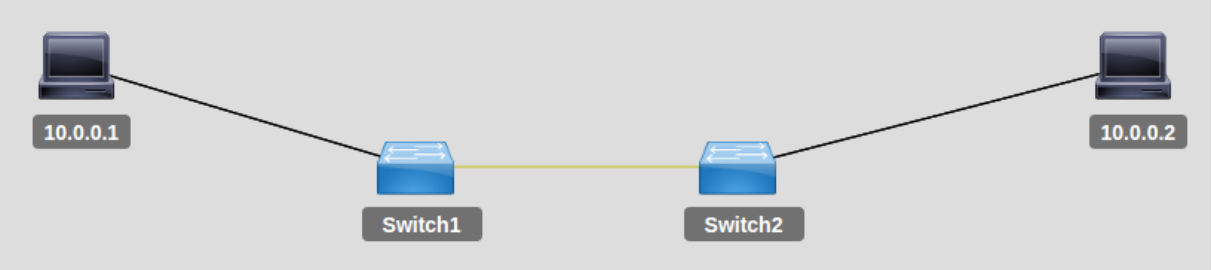
Slika 51. Uspješan *ping* od h1 do h5

```
mininet> h2 ping h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.592 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.117 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.109 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.103 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.112 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.121 ms

--- 10.0.0.4 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5002ms
rtt min/avg/max/mdev = 0.103/0.192/0.592/0.179 ms
```

Slika 52. Uspješan *ping* od h2 do h4

Na slikama 51. i 52. se može uočiti da su *ping*-ovi za relaciju h1 <-> h5 i za relaciju h2 <-> h4 bili uspješni, što znači da je instalacija *Flow Entry*-a preko *OpenDaylight* kontrolera bila pravilna.

Slika 53. prikazuje topologiju za drugi primjer. Blokirati će se TCP protokol prema jednom web poslužitelju, kojeg će za ovaj primjer predstaviti *host* h2 s IP adresom 10.0.0.2. Promet će dolaziti od *host*-a h1 s IP adresom 10.0.0.1. Instalacija *Flow Entry*-a će se izvršiti na Switch2.




Slika 53. Topologija korištena za *Flow Entry* blokiranja TCP protokola

Izvor: Izradio autor

Ovo bi se moglo zamisliti kao jedan *in-network Firewall*. Umjesto da se promet blokira na *host*-u koristeći *software Firewall*, blokirati će se već na *switch*-u. Ovaj način se može npr. koristiti na *Core switch*-evima jednog podatkovnog centra kako bi se mreža zaštitila od neželjenog prometa. Prije nego što će se instalirati *Flow Entry*, mora se potvrditi da h1 može slati zahtjeve na port 80 prema h2. Nakon kreiranja topologije u *Mininet*-u, pomoću komande *xterm h1 h2* u CLI-u, otvaraju se dva nova terminala, jedan za h1 i drugi za h2 te se sada mogu pokrenuti komande za svaki *host* posebno. Koristiti će se *Netcat* za simulaciju web poslužitelja i klijenta.

```
h2> nc -lk 10.0.0.2 80
Hello
Hello
Hello

h2> echo $?
0
```

Slika 54. *Host* h2 kao web poslužitelj

```
h1> echo "Hello" | nc 10.0.0.2 80
h1> echo "Hello" | nc 10.0.0.2 80
h1> echo "Hello" | nc 10.0.0.2 80
```

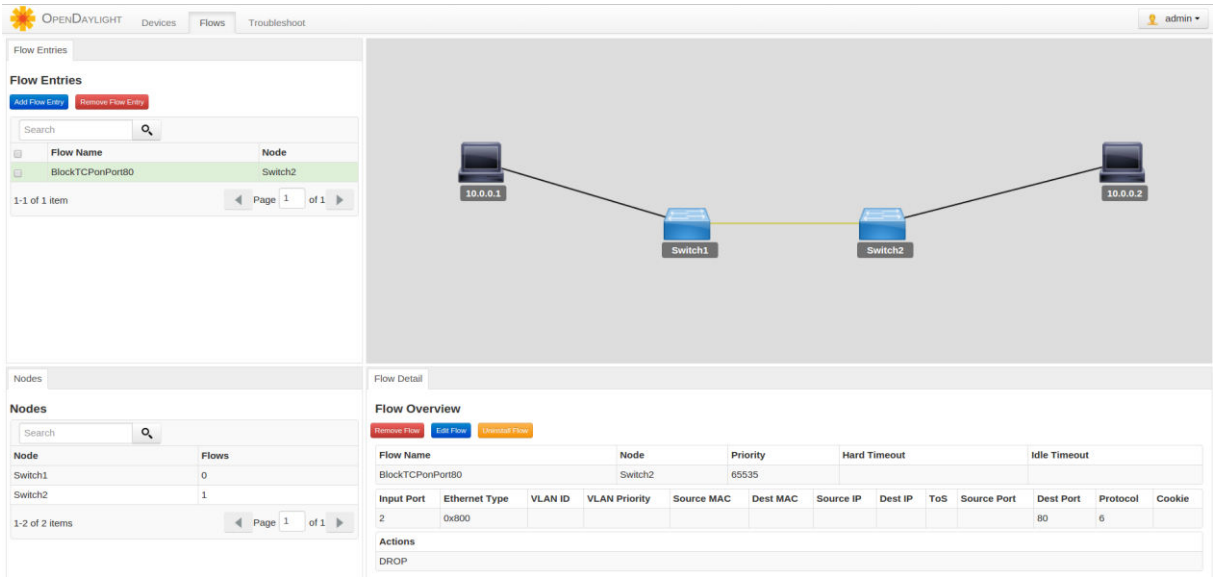
Slika 55. *Host* h1 kao klijent

Na slikama 54. i 55. je vidljivo da je *host* h1 u mogućnosti slati zahtjeve na port 80 prema *host*-u h2. *Host* h2 sluša (opcija -l) na portu 80 za dolazeće zahtjeve i ostaje slušajući zahtjeve (opcija -k). *Host* h1 šalje *string* "Hello" i *host* h2 prikazuje taj *string*, što znači da povezanost radi. Unosom komande *echo $?* se može provjeriti da li *host* h2 može primiti pakete. Ako može, izlazna vrijednost treba biti '0'. Bilo koja druga vrijednost osim '0' znači da ne može.

Sada slijedi instalacija *Flow Entry*-a u *OpenDaylight* kontroleru, a korišteni parametri su sljedeći:

- *Name*: BlockTCPonPort80
- *Node*: Switch2
- *Input Port*: s2.eth2
- *Priority*: 65535
- *Ethernet Type*: 0x800
- *Destination Port*: 80
- *Protocol*: 6 (*Protocol Number* za TCP)
- *Action*: *Drop*

U *OpenDaylight* kontroleru je sada vidljiv instalirani *Flow Entry*, što je prikazano na slici 56.




Slika 56. *Flow Entry* za blokiranje TCP protokola na portu 80

Izvor: Izradio autor

Sada treba provjeriti da li instalirani *Flow Entry* ima učinka, ponovnim slanjem *Hello* poruke preko TCP-a na port 80 kao na slici 57.

```
h2> nc -lk 10.0.0.2 80

h2> echo $?
130
```

Slika 57. Nemogućnost primanja TCP paketa na web poslužitelju

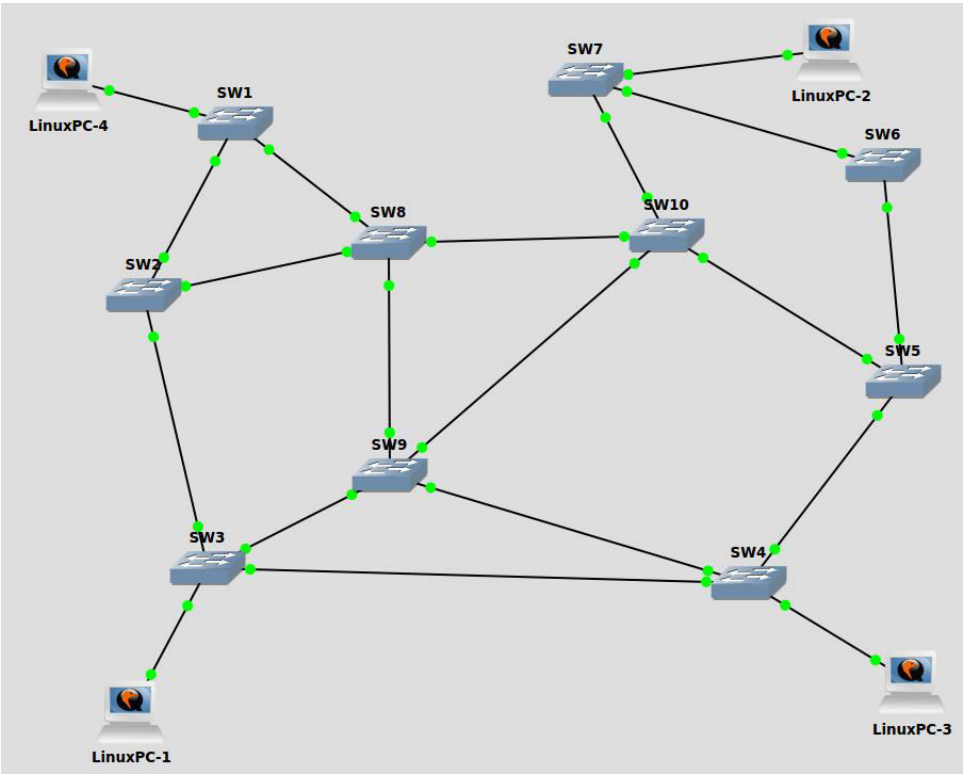
Kao što je vidljivo na slici, *host* h2 ne prikazuje *string* Hello, a izlazna vrijednost komande *echo $?* je '130', tj. nije '0' što znači da *host* h2 više ne može primati TCP pakete na portu 80.

Vidljivo je da se pravila ponašanja jedne SDN mreže mogu na vrlo brzi i jednostavan način mijenjati pomoću kontrolera u stvarnom vremenu. Nije potrebno konfigurirati nekoliko *switch*-eva kao u konvencionalnoj mreži, što dovodi do velike uštede vremena, a i do smanjenja grešaka jer sve obavlja SDN kontroler.

## 7.4. Mjerenje performansi

U poglavlju 6. su analizirani određeni zahtjevi, koje ako se uspješno ispune, mogu omogućiti bolje i kvalitetnije upravljanje mrežom te su navedene prednosti SDN-a u odnosu na konvencionalnu mrežu. Iz mjerenja koje će se provesti se želi vidjeti da li SDN mreža može, uz sve prednosti koje ima u vezi upravljanja mrežom, pružati i dovoljno dobru kvalitetu za određene performanse.

Mjerenje uključuje dva različita scenarija, gdje će se u prvom scenariju mjeriti propusnost i kašnjenje paketa od *client*-a do *servera* pomoću TCP protokola. U drugom scenariju će se mjeriti gubici paketa pomoću UDP protokola. Scenarij 2 se razlikuje u prekidanju linka koji će se izazvati zatvaranjem porta između *switch*-a 1 i *switch*-a 5. Mrežne topologije će se za konvencionalnu i SDN mrežu izraditi u GNS3 emulatoru, kako bi se osigurali isti uvjeti, koristeći stvarni *software* na Cisco *switch*-evima i *Open vSwitch software* na SDN *switch*-evima te koristeći 4 *Linux* PC-a. Slika 58. prikazuje izgled korištene topologije za prvi scenarij.




Slika 58. Topologija za prvi scenarij

Izvor: Izradio autor

Konvencionalna mreža se sastoji od slijedećih uređaja:

- 10 Cisco *c3725 Ethernet Switch Router*
- 4 *Linux* PC (PC-1 i PC-3 su *client*, PC-2 i PC-4 su *server*)

SDN mreža se sastoji od sljedećih uređaja i kontrolera:

- 10 *Open vSwitch 1.11*
- 4 *Linux* PC (PC-1 i PC-3 su *client*, PC-2 i PC-4 su *server*)
- *OpenDaylight* kontroler

Promet će se u trajanju od 5 min generirati pomoću *Distributed Internet Traffic Generatora* (D-ITG) koji je pokrenut na svim PC-ima. Generator će se '"natjerati' da generira 1000 paketa u sekundi. PC-1 i PC-3 su postavljeni kao *Sender*, a PC-2 i PC-4 kao *Receiver*.

Prije samog mjerenja potrebni su određeni koraci koji će omogućiti mjerenje, a to su:

1. Da ne bi došlo do odstupanja u mjerenima, svi PC-jevi bi trebali imati sinkronizirane satove te ih je potrebno spojiti na *"public"* NTP *server*-ima. Zbog toga je na njima potrebno kreiranje *gateway*-a prema Internetu. To će se ostvariti pomoću *Cloud*-a u GNS3 simulatoru koji predstavlja Internet, odnosno *gateway* prema Internetu, na način da se virtualni *Linux* PC (PC-1 do PC-4) spoji na TAP sučelje *host* računala (računalo na kojem se izvodi GNS3 emulator), koje je ustvari "tunel" između *host* računala i virtualnog *Linux* PC-a. Naziv TAP sučelja je za ovaj slučaj tap0. IP adresa na TAP-u *host* računala je ujedno i *default gateway* za virtualni *Linux* PC. Da bi virtualni *Linux* PC još mogao doći do određenih Internet web adresa, potrebno mu je još samo definirati *DNS server* gdje će se izabrati Google DNS (IP adresa je 8.8.8.8). Postupci su vidljivi na slikama 59. i 60. Prikazan je postupak samo za PC-1, za ostale PC-e je potreban isti postupak samo sa IP adresama 10.10.10.3 i 10.0.0.2 (PC-2), 10.10.10.4 i 10.0.0.3 (PC-3) te 10.10.10.5 i 10.0.0.4 (PC-4). *NTP server* prema kojem će se sinkronizirati satovi je CARNet-ov u Zagrebu (Sveučilišni računski centar Sveučilišta u Zagrebu).
2. Kreiranje *Linux bridge*-a: *Open vSwitch* korišten u SDN mreži je *VirtualBox Appliance* u GNS3 emulatoru. Kako bi *switch*-evi mogli komunicirati s *OpenDaylight* kontrolerom, korišteni porti se moraju dodati *Linux bridge*.

Postupak je prikazan za *Open vSwitch3* koji je spojen na PC-1 (slika 62.). Za sve ostale je potreban isti postupak.

```
anthony@anthony:~$ sudo ifconfig tap0 10.10.10.1/24
[sudo] password for anthony:
anthony@anthony:~$
```

Slika 59. Dodavanje IP adrese TAP sučelju na *host* računalu

```
tc@PC-1:~$ sudo hostname PC1
tc@PC-1:~$ sudo ifconfig eth1 10.10.10.2 netmask 255.255.255.0
tc@PC-1:~$ sudo ifconfig eth0 10.0.0.1/8
tc@PC-1:~$ sudo route add default gw 10.10.10.1
tc@PC-1:~$ sudo vi /etc/resolv.conf
nameserver 8.8.8.8
~
~
~
~
I /etc/resolv.conf [Modified] 1/1 100%
```

Slika 60. Dodavanje *gateway*-a i definiranje DNS *server*-a za PC-1

```
tc@PC-1:~$ tce-load -wi ntpclient
tc@PC-1:~$ sudo ntpclient -s -h zg1.ntp.carnet.hr
41621 56358.709   27668.0     27.5  8899131.7   6851.2         0
```

Slika 61. Sinkronizacija sata pomoću CARNet NTP *server*-a

```
tc@vSwitch3:~$ sudo ovs-vsctl add-br br1
tc@vSwitch3:~$ sudo ovs-vsctl add-port br1 eth1
tc@vSwitch3:~$ sudo ovs-vsctl add-port br1 eth2
tc@vSwitch3:~$ sudo ovs-vsctl add-port br1 eth3
tc@vSwitch3:~$ sudo ovs-vsctl add-port br1 eth4
tc@vSwitch3:~$ sudo ovs-vsctl set-controller br1 tcp:192.168.165.1:6633
```

Slika 62. Kreiranje *Linux bridge*-a na *Open* vSwitch3

Nakon uspješnih konfiguracija koje su bile potrebne, pokrenut je generator na PC-1 i PC-3 kao *Sender* te na PC-2 i PC-4 kao *Receiver*. Na slici 63. je prikazan postupak pokretanja generatora na PC-1 kao *Sender*, a na slici 64. je prikazan postupak za pokretanje generatora na PC-2 kao *Receiver*. Isti postupci su potrebni i za PC-3 i PC-4.

```
tc@PC-1:~$ ITGSend -a 10.0.0.2 -c 1024 -C 1000 -T TCP -t 300000
ITGSend version 2.8.1 (r1023)
Compile-time options: dccp bursty multiport
Started sending packets of flow ID: 1
Finished sending packets of flow ID: 1
```

Slika 63. Pokretanje generatora na PC-1

```
tc@PC-2:~$ ITGRecv
ITGRecv version 2.8.1 (r1023)
Compile-time options: dccp bursty multiport
Press Ctrl-C to terminate
Listening on TCP port : 8999
Finish on TCP port : 8999
```

Slika 64. Pokretanje generatora na PC-2

Mjerenje je trajalo 5 minuta te su rezultati vidljivi na slikama 65. i 66.

```
****************  TOTAL RESULTS   ******************
_____________________________________________________
Number of flows          =             2
Total time               =    299.994927 s
Total packets            =        209253
Minimum delay            =      0.001536 s
Maximum delay            =      0.069049 s
Average delay            =      0.004798 s
Average jitter           =      0.001250 s
Delay standard deviation =      0.001764 s
Bytes received           =     214275072
Average bitrate          =   5714.098545 Kbit/s
Average packet rate      =    697.521795 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =             0 pkt
Error lines              =             0
```

Slika 65. Rezultati prvog mjerenja za konvencionalnu mrežu

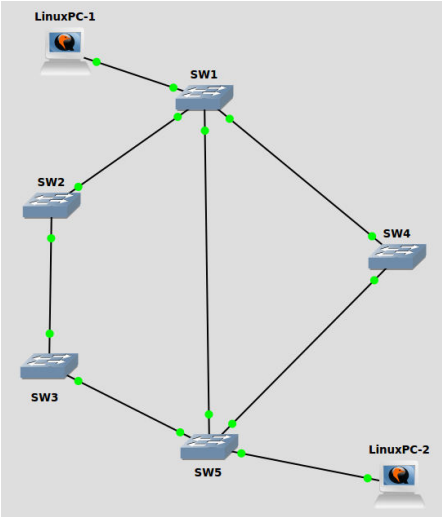
```
****************  TOTAL RESULTS   ******************
_____________________________________________________
Number of flows          =             2
Total time               =    299.985785 s
Total packets            =        222052
Minimum delay            =      0.001964 s
Maximum delay            =      0.076240 s
Average delay            =      0.004424 s
Average jitter           =      0.001042 s
Delay standard deviation =      0.001785 s
Bytes received           =     227381248
Average bitrate          =   6063.787269 Kbit/s
Average packet rate      =    740.208407 pkt/s
Packets dropped          =             0 (0.00 %)
Average loss-burst size  =             0 pkt
Error lines              =             0
```

Slika 66. Rezultati prvog mjerenja za SDN mrežu

Rezultati za prvo mjerenje pokazuju, da ukoliko su identični uvjeti, SDN rješenje može pružati u najmanju ruku iste performanse. Vrlo mala razlika u brojkama je zbog toga što znajući topologiju unaprijed, prvi paketi će se na početku kod SDN rješenja prije poslati nego u konvencionalnoj mreži. Razlika je zbog toga što *switch*-evi prvo moraju naučiti topologiju i tu se stvara kašnjenje u odnosu na SDN. Iako su bili identični uvjeti i korišteni su stvarni *software*-i, ovi rezultati se u stvarnosti mogu naravno razlikovati i varirati jer utječu razni faktori kao što su udaljenost, linkovi i naravno korišteni *hardware*-a. *Switch* koji košta nekoliko 1000$ će zasigurno imati bolje specifikacije od *switch*-a koji košta nekoliko 100$.

Vrlo je teško usporediti konvencionalnu mrežu s SDN-om na temelju izmjerenih performansi, jer je SDN dizajniran s ciljem postizanja prednosti iz 6. poglavlja, a to je fleksibilnije i lakše upravljanje mrežama. Ujedno se SDN rješenja razlikuju od dobavljača do dobavljača i sama mreža se može konfigurirati po potrebi. Sve ovisi za što organizacija želi koristiti mrežu te koje usluge želi pružati i po tome će se performanse mreže prilagoditi. No ako se uzme u obzir da uz dobivene rezultate u najmanju ruku pruža iste ili slične performanse, ako se radi o istim uvjetima i vrsti mreže te dodaju prednosti koje su navedene, može se zaključiti da SDN može unaprijediti poslovanje jedne tvrtke.

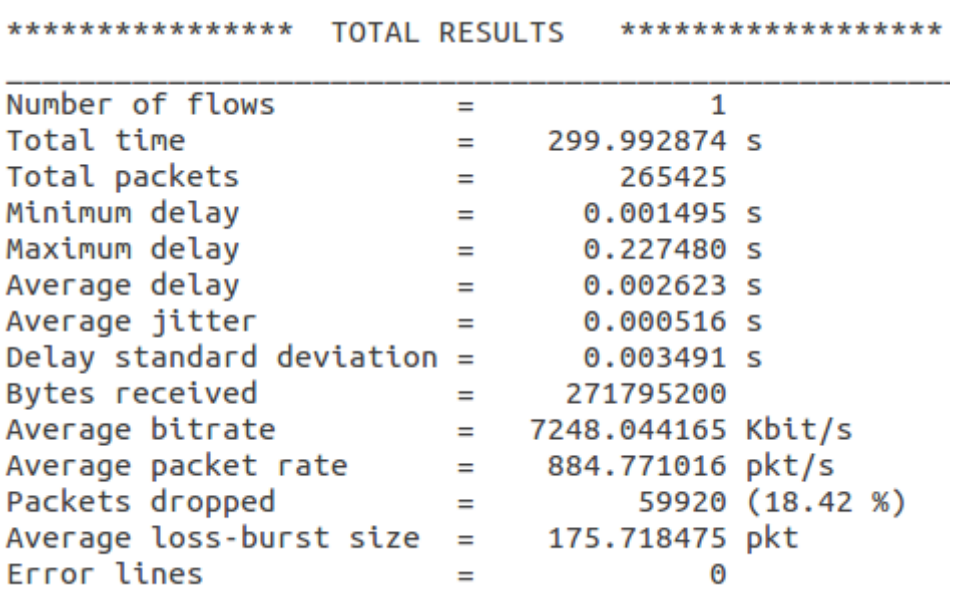
Topologija korištena za drugi scenarij je prikazana na slici 67. Postavke su iste kao i u prvom scenariju, samo što su jedan *Sender* (PC-1) i jedan *Receiver* (PC-2).



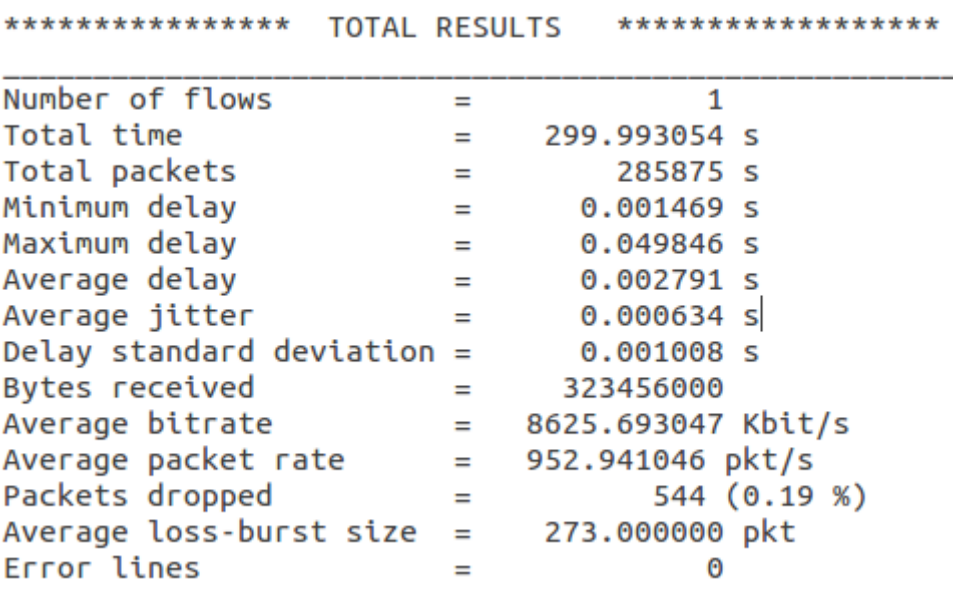

Slika 67. Topologija za drugi scenarij

Izvor: Izradio autor

Mjerenje je također trajalo 5 minuta te su rezultati vidljivi na slikama 68. i 69.




Slika 68. Rezultati drugog mjerenja za konvencionalnu mrežu




Slika 69. Rezultati drugog mjerenja za SDN mrežu

Drugo mjerenje prikazuje dosta veću razliku u rezultatima. Dok su za SDN gubitci bili samo 0.19%, za konvencionalnu mrežu su bili 18.42%. Naravno da i ovdje u stvarnosti razlika može biti manja ili veća, no SDN rješenje će i u stvarnosti imati bolji rezultat za ovakav slučaj mjerenja. Poznavajući cijelu topologiju, kontroler točno zna kuda treba paket ići ukoliko dođe do kvara linka ili zatvaranja određenog porta i kao što je vidljivo po rezultatima, obavlja to vrlo brzo. U konvencionalnim *switch-*

evima, topologija mora biti prvo ponovno naučena, jer *switch* vidi prvog do sebe i ne zna stanje mreže napamet. Kada dođe do kvara linka ili zatvaranja porta, konvergencija STP protokola traje određeno vrijeme:

- *Hello time* = 2 s
- *Max Age* = 10 x *Hello time*
- *Listening* = 15 s
- *Learning* = 15 s

STP ima 4 stanja, a to su *blocking*, *listening*, *learning* i *forwarding*. Jednom kada je port blokiran, u tom stanju ostaje 20 sekundi. Onda idućih 15 sekundi bude u stanju slušanja te onda 15 sekundi bude u stanju učenja. Ako se zbroje sva stanja skupa s *Hello* vremenom dođe se do brojke od 52 sekunde. Zbog toga je tolika razlika u mjerenjima jer *switch*-evi moraju ponovno naučiti topologiju, što kod SDN-a nije potrebno i tu se vidi dodatna prednost SDN-a.

# 8. ZAKLJUČAK

Softverski definirana mreža obuhvaća arhitekturu koja je dinamična, ekonomična i prilagodljiva te se lako rukuje njome, što ju čini idealnom za dinamičnu prirodu današnjih aplikacija. Odvajanje kontrolne i podatkovne cjeline omogućuje da kontrola mreže postaje izravno programabilna i da infrastruktura nižeg sloja bude odvojena za aplikacije i mrežne usluge. SDN nudi centralizirani pogled na mrežu, dajući SDN kontroleru zbog kontrolne cjeline kojom upravlja sposobnost da djeluju kao "mozak" mreže te je strateška kontrolna točka u SDN mreži, koji sa *switch*-evima/*router*-ima komunicira putem *southbound* API-ja, dok s aplikacijama komunicira putem *northbound* API-ja. Jedan od najpoznatijih protokola koji se koristi za komunikaciju između kontrolera i mrežnih uređaja je *OpenFlow*.

Centralizirana, programabilna SDN okruženja mogu se lako prilagoditi brzo promjenjivim potrebama tvrtki. Ključne prednosti SDN-a zbog svoje arhitekture i odvajanja cjelina su agilnost i fleksibilnost. SDN omogućuje organizacijama brzi razvoj novih aplikacija, usluga i infrastruktura kako bi se brzo zadovoljili promjenjivi poslovni ciljevi, fleksibilan odabir načina i djelovanja mreže. Zbog odvajanja cjelina moguće je eksperimentirati s mrežnom konfiguracijom bez da se utječe na samu mrežu te brže implementiranje novih inovacija i usluga. Ponašanje mreže se može u svakom trenutku prilagoditi potrebama sa jednog centraliziranog mjesta. Zbog odvajanja *software*-a na samim *switch*-evima, nije potrebna nabava skupih mrežnih uređaja i moguće je koristiti opremu od različitih dobavljača.

Zbog različite arhitekture u odnosu na konvencionalnu mrežu, SDN mreža pruža nove mete napada. Glavna meta je sami kontroler, jer se njima upravlja cijelom mrežom te *southbound* i *northbound* API-ji. Odgovarajućim metodama zaštite za API-je i kontroler te korištenjem sigurnosnih aplikacija, SDN može povećavati sigurnost. Razlog je centralizirani pogled na cijelu mrežu što omogućuje brzo otkrivanje nepravilnih ponašanja i brzo djelovanje u stvarnom vremenu.

Implementiranje SDN rješenja zahtjeva dobro planiranje. Organizacije trebaju imati jasnu ideju o prednostima za koje misle da će ih realizirati implementacijom SDN-a. U mnogim slučajevima, softverski definirano rješenje ne mora nužno izgledati drugačije od konvencionalne mreže te se SDN rješenja razlikuju od dobavljača do

dobavljača. Ključni izazovi pri implementaciji i konfiguriranju SDN rješenja za administratore predstavljaju performanse mreže, skalabilnost, sigurnost i interoperabilnost.

Prilikom konfiguracija mreža mogu se uočiti bitne razlike između SDN mreže i konvencionalne mreže. U konvencionalnoj mreži potrebno je konfigurirati ručno svaki *switch* posebno što zahtijeva dosta više postupaka i vremena. Ukoliko se mreža sastoji od tisuća mrežnih uređaja i *host*-ova, što je česti slučaj u današnjim mrežama, svaki *switch* se za odgovarajući promet, *flow* i svaku promjenu *flow*-a mora posebno konfigurirati, što naravno povećava broj postupaka i vrijeme potrebno za to. Ako se još dodaju VLAN-ovi i QoS, posao administratora se dodatno povećava, jer se potrebna konfiguracija mora napraviti na svakom *switch*-u, a povećava se i mogućnost pogreške. U SDN rješenju sve navedene postupke, od konfiguriranja *switch*-eva i učenja mrežne topologije, prilikom prvog spajanja s *switch*-evima obavlja SDN kontroler s jednog centraliziranog mjesta i to u vrlo kratkom roku te u tome leži velika prednost u odnosu na konvencionalne mreže. *Switch* mora biti spojen na SDN kontroler te sve ostalo obavlja sam kontroler. Prednost učenja cjelokupne topologije i pogled na cijelu mrežu vidljiva je na temelju rezultata za scenarij 2 gdje je velika razlika u izgubljenim paketima u odnosu na konvencionalnu mrežu. Prema analizi i istraživanju, rezultatima mjerenja te uzimajući u obzir sve prednosti SDN-a se može zaključiti da SDN može unaprijediti poslovanje jedne organizacije (tvrtke).

# LITERATURA


[1] Shukla, V.: "Introduction to Software Defined Networking - OpenFlow & VxLAN", CreateSpace Independent Publishing Platform, Kindle Edition, 2013.

[2] Nadeau, T. D., Gray, K.: "SDN: Software Defined Networks", O'Reilly Media, Inc., 1. izdanje, California, 2013.

[3] Cisco: "Software-Defined Networking: Why We Like It and How We Are Building On It", White Paper, 2013.

[4] Sundararajan, R. K.: "Software Defined Networking (SDN) - A definitive guide", Kloudspunn Press, Kindle Edition, 2013.

[5] Open Network Foundation: "Software-Defined Networking: The New Norm for Networks", White Paper, 2012.

[6] Tiwari, V.: "SDN and OpenFlow for beginners with hands on labs", M.M. D.D. Multimedia LLC., Kindle Edition, Northville, 2013.

[7] Göransson, P., Black, C.: "Software Defined Networks - A Comprehensive Approach", Elsevier, Inc., Waltham, 2014.

[8] Open Network Foundation: "OpenFlow Switch Specification Version 1.0", 2009.

[9] Brandt, M., Khondoker, R., Marx, R., Bayarou, K.: "Security Analysis of Software Defined Networking Protocols — OpenFlow, OF-Config and OVSDB" in Proceedings of the AINTEC 2014 on Asian Internet Engineering Conference, 2014

[10] Benton, K., Camp, L. J., Small, C.: “OpenFlow Vulnerability Assessment” in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN ’13. ACM, August 2013, pp. 151–152

[11] Kreutz, D., Ramos, F. M., Verissimo, P.: “Towards Secure and Dependable Software-Defined Networks” in Proceedings of the Second ACM SIGCOMM

Workshop on Hot Topics in Software Defined Networking, ser. HotSDN ’13. ACM, August 2013, pp. 55–60

[12] Hartman, S., Wasserman, M., Zhang, D.: “Security Requirements in the Software Defined Networking Model”, Internet Engineering Task Force, Internet-Draft draft-hartman-sdnsec-requirements-01, April 2013

[13] Klöti, R., Kotronis, V., Smith, P.: “OpenFlow: A Security Analysis” in Proceedings of the 8th Workshop on Secure Network Protocols (NPSec), part of IEEE ICNP, ser. NPSec ’13. IEEE, October 2013

[14] Sezer, S., Scott-Hayward, S., Pushpinder Kaur, C.: “Are we ready for SDN? Implementation challenges for software-defined networks” IEEE Communications Magazine, vol. 51, no. 7, July 2013, pp. 36–43

[15] Shin, S., Yegneswaran, V., Porras, P., Gu, G.: “AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks” in Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ser. CCS ’13. ACM, November 2013, pp. 413–424

[16] Wen, X., Chen, Y., Hu, C., Shi, C., Wang, Y.: “Towards a Secure Controller Platform for OpenFlow Applications” in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN ’13. ACM, August 2013, pp. 171–172

[17] Shirali-Shahreza, S., Ganjali, Y.: “Empowering Software Defined Network controller with packet-level information” in Proceedings of the 2013 IEEE International Conference on Communications Workshops, ser. ICC ’13. IEEE Computer Society, June 2013, pp. 1335–1339

[18] Shirali-Shahreza, S., Yashar, G.: “Efficient Implementation of Security Applications in OpenFlow Controller with FleXam” in Proceedings of the 21st Annual Symposium on High-Performance Interconnects, ser. HOTI ’13. IEEE Computer Society, August 2013, pp. 49–54

[19] Shirali-Shahreza, S., Ganjali, Y.: “FleXam: flexible sampling extension for monitoring and security applications in openflow” in Proceedings of the

Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN ’13. ACM, August 2013, pp. 167–168

[20] Shin, S., Gu, G.: “CloudWatcher: Network Security Monitoring Using OpenFlow in Dynamic Cloud Networks” in Proceedings of the 2012 20th IEEE International Conference on Network Protocols (ICNP), ser. ICNP ’12. IEEE Computer Society, October 2012, pp. 1–6

[21] Braga, R., Mota, E., Passito, A.: “Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow” in Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks, ser. LCN ’10. IEEE Computer Society, October 2010, pp. 408–415

[22] Shin, S., Gu, G.: "Attacking Software-defined Networks: A First Feasibility Study” in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN ’13. ACM, August 2013, pp. 165–166

[23] Son, S., Shin, S., Yegneswaran, V., Porras, P., Gu, G.: “Model Checking Invariant Security Properties in OpenFlow” in Proceedings of IEEE International Conference on Communications, ser. ICC ’13. IEEE Computer Society, June 2013, pp. 1974–1979

[24] Shin, S., Porras, P., Yegneswaran, V., Fong, M., Gu, G., Tyson, M.: “FRESCO: Modular Composable Security Services for Software Defined Networks” in Proceedings of the 20th Annual Network and Distributed System Security Symposium, ser. NDSS ’13. The Internet Society, February 2013

[25] Shin, S., Porras, P., Yegneswaran, V., Gu, G.: “A Framework For Integrating Security Services into Software-Defined Networks” in Proceedings of the 2013 Open Networking Summit (Research Track poster paper), ser. ONS ’13, April 2013

[26] Metzler, J., Metzler, A.: "The 2013 Guide to Network Virtualization and SDN", Webtorials Analyst Division, 2013.

[27] Vissicchio, S., Vanbever, L., Bonaventure, O.: "Opportunities and Research Challenges of Hybrid Software Defined Networks" in Proceedings of the 13th ACM Workshop on Hot Topics in Networks, 2014, pp. 70-75

[28] Kepes, B.: "SDN meets the real-world: Implementation benefits and challenges", GIGAOM RESEARCH, 2014.

[29] Agarwal, S., Kodialam, M., Lakshman, T. V.: "*Traffic Engineering in Software Defined Networks*" in Proceedings of IEEE INFOCOM, Turin 2013, pp. 221-2219

[30] Nascimento, M. R., Rothenberg, C. E., Salvador, M. R., Correa, C. N. A., de Lucena, S. C., Magalhaes, M. F.: "*Virtual Routers as a Service: the RouteFlow approach leveraging Software-Defined Networks*", in Proceedings of the 6th International Conference on Future Internet Technologies, New York 2011, pp. 34-34

[31] Shin, M.K., Nam, K.-H., Kim, H. J.: "*Software-defined networking (SDN): A reference architecture and open APIs*" Proceedings, 2012 International Conference on ICT Convergence (ICTC), 15–17 October, 2012, pp.360–361

[32] Big Switch Networks: "*The Open SDN Architecture*", White Paper, 2013.

[33] CITRIX Systems: "*SDN 101: An Introduction to Software Defined Networking*", White Paper, 2014.

[34] Open Network Foundation: "*SDN Architecture Overview*", White Paper, 2013.

[35] Open Network Foundation: "*SDN Architecture Overview*", White Paper, 2014.

[36] Open Network Foundation: "*SDN Architecture*, *Issue 1*", White Paper, 2014.

[37] Open Network Foundation: "*OpenFlow Switch Specification Version 1.1*", 2011.

[38] Open Network Foundation: "*OpenFlow Switch Specification Version 1.2*", 2011.

[39] Open Network Foundation: "*OpenFlow Switch Specification Version 1.3*", 2012.

[40] Open Network Foundation: "*OpenFlow Switch Specification Version 1.4*", 2013.

[41] Open Network Foundation: "*OpenFlow Switch Specification Version 1.5*", 2014.

[42] Open Network Foundation: "*Migration Use Cases and Methods*", Migration Working Group

[43] YouTube video materijal - *Mininet Installation Instructions*: https://www.youtube.com/watch?v=yNmv7GiHIKE (24.7.2015.)

[44] YouTube video materijal - *Introduction to Mininet*: https://www.youtube.com/watch?v=jmlgXaocwiE (24.7.2015.)

[45] YouTube video materijal - *Mininet Basics Tutorial - Essentials You Need to Know*: https://www.youtube.com/watch?v=oPtVYSyN1wE (24.7.2015.)

[46] YouTube video materijal - *Mininet Custom Topologies*: https://www.youtube.com/watch?v=yHUNeyaQKWY (25.7.2015.)

[47] YouTube video materijal - *OpenDaylight Tutorial for Developers and Users*: Overview | OpenDaylight Summit 2014: https://www.youtube.com/watch?v=g_Gp30kGdec&list=PL8F5jrwEpGAiMH3is6WaQJXaZ26AefPhU (27.7.2015.)

[48] YouTube video materijal - *User Tutorial: The Basic Operations Guide for OpenDaylight*: https://www.youtube.com/watch?v=P7tmeIDNHJ0 (28.7.2015.)

[49] YouTube video materijal - *OpenDaylight OpenFlow Controller Tutorial*: https://www.youtube.com/watch?v=hXWhgWMoNHM (28.7.2015.)

[50] YouTube video materijal - *GNS3 tutorial install and configure Dynamips in Ubuntu*: https://www.youtube.com/watch?v=PRmBu4Lc-B0 (14.8.2015.)

[51] YouTube video materijal - *GNS3 -- How to set up switching in GNS3*: https://www.youtube.com/watch?v=f868S9NnmHg (14.8.2015.)

[52] YouTube video materijal - *How to emulate a Cisco Switch in GNS3*: https://www.youtube.com/watch?v=2gQcLDSj-_c (15.8.2015.)

## POPIS KRATICA I AKRONIMA

API - *Application Programming Interface*

ARP - *Address Resolution Protocol*

BGP - *Border Gateway Protocol*

CLI - *Command-Line Interface*

DDoS - *Distributed Denial-of-Service Attack*

DNS - *Domain Name System*

DoS - *Denial-of-Service Attack*

D-ITG - *Distributed Internet Traffic Generator*

FIB - *Forwarding Information Base*

GNS3 - *Graphical Network Simulator 3*

GUI - *Graphical User Interface*

ICMP - *Internet Control Message Protocol*

IEEE - *Institute of Electrical and Electronics Engineers*

IP - *Internet Protocol*

JSON - *JavaScript Object Notation*

LLDP - *Link Layer Discovery Protocol*

MAC - *Media Access Control*

MITM - *Man in the Middle attack*

OF-Config - *OpenFlow Configuration and Management Protocol*

ONF - *Open Networking Foundation*

OS - *Operating System*

OSPF - *Open Shortest Path First*

OVSDB - *Open vSwitch Database*

PC - *Personal Computer*

QoS - *Quality of Service*

REST - *REpresentational State Transfer*

RIB - *Routing Information Base*

SDN - *Software Defined Network*

SSH - *Secure Shell*

STP - *Spanning Tree Protocol*

TCP - *Transmission Control Protocol*

TLS - *Transport Layer Security*

UDP - *User Datagram Protocol*

VLAN - *Virtual Local Area Network*

VM - *Virtual Machine*

VxLAN - *Virtual Extensible LAN*

XML - *Extensible Markup Language*

# POPIS SLIKA I TABLICA

## *Popis slika*

***Popis tablica***