UDC: 004.4 Info M str. 34-39

SOFTVERSKI DEFINISANE MREŽE – SIMULACIJA VIRTUELNE RAČUNARSKE MREŽE UPOTREBOM MININET OKRUŽENJA
SOFTWARE DEFINED NETWORKING – SIMULATION OF VIRTUAL NETWORKS USING MININET ENVIRONMENT

MSc, Bojan Marčeta, Miloš Živadinović

REZIME: U radu je prikazan novi koncept upravljanja računarskim mrežama, Softverski definisane mreže - osnovna ideja, arhitektura, osnovni pojmovi, prednosti i nedostaci. Predstavljeno je i simulaciono okruženje Mininet za kreiranje i testiranje virtuelnih topologija, u kome je prikazan postupak rada sa virtuelnim uređajima, hostovima i kontrolerima. Opisan je i postupak povezivanja Mininet okruženja sa eksternim kontrolerom upotrebom Opendaylight projekta.

KLJUČNE REČI: SDN, OpenFlow, Mininet, Opendaylight

ABSTRACT: This paper presents a new concept of managing computer networks, Software defined networking - the basic idea, architecture, basic concepts, advantages and disadvantages. It is also presented Mininet, simulation environment for creating and testing virtual topology, showing the process of working with virtual devices, hosts and controllers. It is also described the procedure for connecting Mininet environment with an external controller using Opendaylight project.

KEY WORDS: SDN, OpenFlow, Mininet, Opendaylight

1. UVOD

Razvojem informacionih tehnologija na tržištu se neprekidno javlja potreba za inovacijama i unapređenjem trenutnog stanja informacionih sistema. Potreba za brzim uvođenjem novih usluga i migracijom korisnika dovela je do zahteva koje konvencionalne računarske mreže ne mogu da ispune. Standardne računarske mreže su nefleksibilne, sve ozbiljnije promene su veoma vremenski zahtevne, što dovodi do toga da računarske mreže, kakvim ih danas poznajemo, nisu u stanju da odgovore na zahteve tržišta [1]. Upravo iz tog razloga, nastaje jedna u potpunosti nova paradigma, a to su softverski definisane računarske mreže.

2. KONVENCIONALNE RAČUNARSKE MREŽE

Današnje računarske mreže uglavnom zavise od potreba i veličine organizacije, i podređene su pružanju određenih servisa krajnjem korisniku ili grupama korisnika.

Prilikom kreiranja mrežne infrastrukture potrebno je voditi računa da mreža bude hijerarhijski organizovana, da je fleksibilna i modularna. U većim mrežama prisutna je tzv. troslojna arhitektura koja podrazumeva da postoje tri sloja (*core*, *distribution* i *access*). *Access* (pristupni) sloj čine uređaji preko kojih se krajnji korisnici spajaju na mrežu. Distributivni sloj obezbeđuje redundansu i sakuplja saobraćaj sa nižeg sloja, i prosleđuje ga ka *core* sloju, koji predstavlja *backbone* same mreže.

Najveći problem trenutno je što se proces upravljanja podacima (*control plane*) i proces prosleđivanja podataka (*data plane*) nalaze u okviru istog fizičkog uređaja. Upravljanje mrežom postaje veliki izazov kada se koriste uređaji različitih proizvođača iz razloga moguće neinteroperabilnosti, različitih rešenja, *proprietary* rešenja, nestandardizovanih protokola i korisničkih interfejsa. Neke funkcionalnosti su implementirane u sam hardver. Sve ovo dovodi do toga da je ovakav pristup veoma skup za održavanje, nije fleksibilan, i ne može uvek da odgovori na novonastale potrebe za resursima i uslugama.

Dodavanje novih uređaja, uklanjanje postojećih, pisanje i implementacija globalnih sigurnosnih polisa, su veoma kompleksne i vremenski zahtevne radnje, koje moraju da se obavljaju ručno, pa mogu izazvati u nekim slučajevima i privremeni prekid u pružanju neke usluge. Ovakav statički pristup obeshrabruje svaki pokušaj promene u konfiguraciji same mreže, i upravo iz tog razloga predstavlja ograničenje u odgovoru na zahteve tržišta i novih aplikacija, koji uglavnom zahtevaju dinamičke i brze promene.

Kako bi se upravljanje olakšalo, ideja je da se koristi hardver sa standardizovanim softverom i grafičkim interfejsom, nezavisno od proizvođača opreme. Na taj način postigla bi se optimalna podela poslova između hardvera i softvera, lakše bi se dodeljivali resursi u zavisnosti od potreba korisnika, i brže bi se implementirali novi servisi.

3. SOFTVERSKI DEFINISANE MREŽE

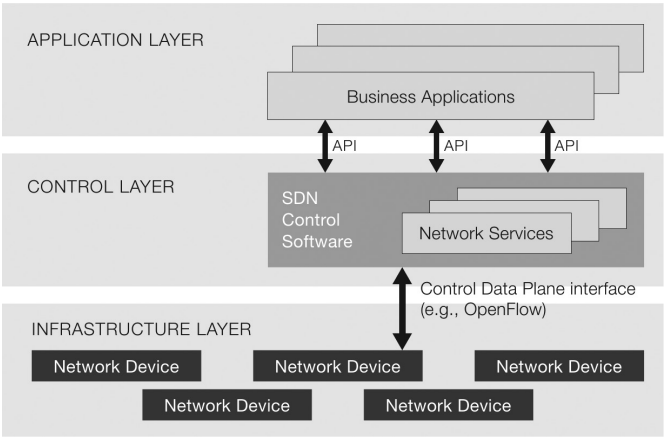
SDN (*Software Defined Networking*) predstavlja novi pristup u upravljanju računarskim mrežama. Osnovna ideja je da se odvoje procesi prosleđivanja i upravljanja (*control* i *data plane*), na način što će se upravljanje centralizovati, a niži slojevi mrežne infrastrukture će biti apstrahovani. Dve osnovne karakteristike na kojima se zasnivaju SDN su programabilnost i apstrakcija.

Programabilnost znači da se mrežama upravlja putem softvera. Korišćenjem API-ja otvorenog standarda moguće je implementirati aplikacije na način da se u zavisnosti od samih potreba, mrežni resursi dinamički (automatski) dodeljuju, bez obzira na proizvođače samog hardvera.

Apstrakcija predstavlja izdvajanje korisničkih aplikacija od mrežnih elemenata, koji su apstrahovani u odnosu na upravljački sloj. Na taj način aplikacije nisu svesne mrežne infrastrukture koja se nalazi ispod [2].

3.1. *Arhitektura SDN*

Arhitektura SDN se u osnovi može predstaviti iz tri dela: aplikativni sloj (korisničke aplikacije), kontrolni sloj i infrastrukturni sloj (mrežni elementi).

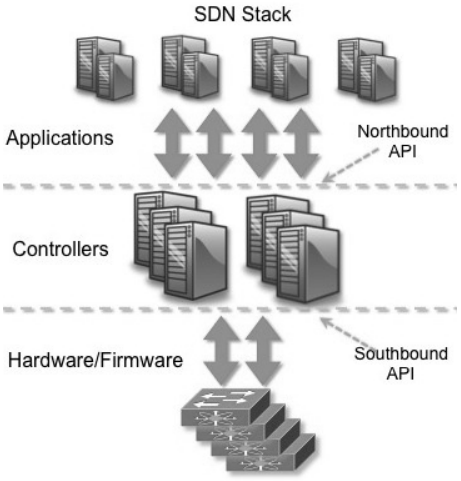


Slika 1. *Arhitektura SDN [3]*

Upravljanje računarskom mrežom je u ovom slučaju centralizovano i vrši se preko SDN kontrolera, koji se sa strane aplikacija vidi kao jedan logički svič. Ovakav pristup omogućava administratorima da programiraju, upravljaju, obezbeđuju i optimizuju mrežu preko automatizovanog SDN softvera, koji mogu i sami da napišu, jer ne zavisi od *proprietary* rešenja [4].

Ključna stvar u implementaciji SDN mreže je upotreba otvorenih standarda. Na taj način se pojednostavljuje i dizajn i samo upravljanje mrežom, jer instrukcije ne zavise od različitih proizvođača opreme i njihovih rešenja i standarda, nego dolaze od SDN kontrolera.

Za komunikaciju između aplikativnog i kontrolnog sloja koristi se tzv. *Northbound API*, dok je *Southbound API* zadužen za komunikaciju između kontrolnog i sloja infrastrukture. Karakteriše ih da su implementirani u otvorenom standardu i nezavisni su od proizvođača opreme. Takođe, u potpunosti su interoperabilni [5].



Slika 2. *SDN apstrakcija [6]*

4. OPENFLOW PROTOKOL

OpenFlow predstavlja prvi i jedini u potpunosti nezavisan standard za komunikaciju između kontrolnog i sloja infrastrukture u SDN arhitekturi. Razvijen je od strane ONF (*Open Networking Foundation*). OpenFlow omogućava direktan pristup i upravljanje procesom prosleđivanja podataka (*data plane*) na uređajima kao što su ruteri i svičevi, bilo fizički ili virtuelni.

Prva verzija protokola (verzija 1.1) se pojavila u februaru 2011. godine. Trenutno je aktuelna verzija 1.4. ONF za cilj ima promociju SDN arhitekture i samog OpenFlow protokola kao standarda. U ovom projektu ONF ima podršku velikih proizvođača, poput *Alcatel-Lucent, Cisco, Huawei, Dell, HP, Juniper, Brocade* i mnogih drugih, koji u svojim proizvodima imaju podržan OpenFlow protokol. OpenFlow predstavlja pragmatičan kompromis: s jedne strane, omogućava istraživačima da eksperimente izvode na jedinstven način sa heterogenim hardverom; dok sa druge strane, proizvođači opreme ne treba da otkrivaju unutrašnju organizaciju i rad svojih uređaja.

Danas je skoro nemoguće testirati nove ideje, protokole i mrežne arhitekture u realnom okruženju. Za to postoji više razloga. Pre svega, teško je obezbediti dovoljno veliku infrastrukturu koja bi bila relevantna za testiranje novih ideja. U slučaju ako se testiranje izvodi na produkcionoj mreži teško je izolovati testni saobraćaj od stvarnog, a pored toga može doći i do narušavanja funkcionalnosti same mreže.

Jedna od mogućnosti je da se koristi tzv. *testbed*, poput *Geni* [7], *Deterlab, Emulab* i sl. Geni predstavlja testno okruženje koje omogućava testiranje novih protokola, mrežnih koncepata i sl., u realnom okruženju. Druga opcija je upotreba SDN-a i OpenFlow protokola.

Većina modernih rutera i svičeva upravlja paketima na osnovu tabela tokova (*flow tables*), koje se pak razlikuju u zavisnosti od samog proizvođača. Međutim, u svoj toj različitosti moguće je identifikovati veliki skup instrukcija koje su zajedničke za mnoge uređaje. Upravu tu činjenicu koristi OpenFlow protokol, koji predstavlja komunikacioni protokol ka mrežnim uređajima preko koga je moguće menjati pravila u samim tabelama tokova.

Ključne komponente svakog OpenFlow sviča su: (1) tabela toka, u kojoj je svaki unos povezan sa određenom akcijom, koja govori sviču kako da procesira tok; (2) sigurni komunikacioni kanal, koji povezuje svič sa kontrolerom i omogućava razmenu paketa i komandi.

Tipična tabela toka se sastoji od tri polja: (1) pravila za prosleđivanje paketa; (2) akcije koja se vezuje za dati tok podataka; (3) polja za statistiku, kao što je broj obrađenih paketa iz datog toka.

Tok može da bude TCP konekcija, MAC adresa ili IP adresa, ili svi paketi sa određene MAC ili IP adrese, VLAN tag ili port sa koga paket dolazi. Svaki tok ima akciju koja se vezuje za njega. Postoje tri osnovne akcije koje moraju da postoje. To su: (1) prosleđivanje toka na određeni port ili portove, što omogućava paketu da bude rutiran kroz mrežu; (2) enkapsuliranje i prosleđivanje paketa toka ka kontroleru, koje se obično

obavlja za prvi paket u novom toku, kako bi kontroler odlučio da li je potrebno dodati tok u tabelu toka; (3) odbacivanje (*drop*) paketa [8].

OpenFlow kanal predstavlja interfejs koji povezuje OpenFlow svič sa kontrolerom. Kroz ovaj interfejs kontroler upravlja svičom, prima obaveštenja od sviča i šalje poruke ka njemu. Poruke koje se šalju ovim kanalom moraju biti formatirane u skladu sa OpenFlow standardom. OpenFlow kanal se obično kriptuje putem TLS-a, ali može i da se pokreće direktno preko TCP-a.

4.1. *Vrste poruka*

OpenFlow protokol podržava tri vrste poruka koje se razmenjuju. To su: (1) *controller-to-switch*, (2) asinhrone i (3) simetrične. *Controller-to-switch* poruke se iniciraju direktno od strane kontrolera i služe za upravljanje i proveru stanja sviča. Asinhrone poruke generiše svič kako bi obavestio kontroler o događajima u mreži i stanju sviča. Simetrične poruke mogu generisati i svič i kontroler i mogu se slati u oba smera. To su poruke tipa *hello* prilikom uspostave same konekcije, ili poruke tipa *echo request/reply*.

4.2. *Uspostava konekcije*

Prilikom povezivanja i paljenja uređaja svič mora da bude u stanju da ostvari komunikaciju sa kontrolerom preko IP adrese i određenog porta koje je podesio korisnik. Ako su ovi parametri podešeni svič inicira otvaranje standardne TLS ili TCP konekcije ka kontroleru. Kontroler prihvata nove konekcije na portu 6633. Svič i kontroler se međusobno autentifikuju razmenom sertifikata.

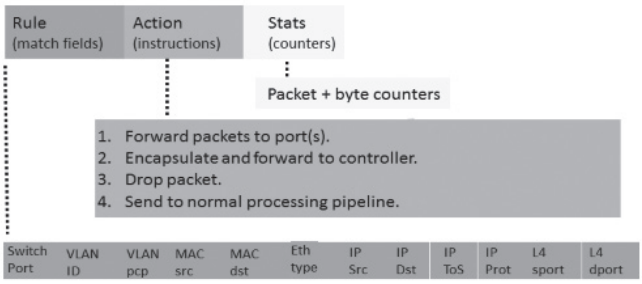
Buduće verzije protokola će imati opisanu specifikaciju za DCDP (*Dynamic Controller Discovery Protocol*) kako bi se IP adresa i port za komunikaciju definisali automatski za vreme paljenja uređaja. Kada se konekcija uspostavi, svaka strana koja učestvuje u komunikaciji šalje **OFPT_HELLO** poruku sa verzijom OpenFlow protokola koji podržava. Nakon razmene ovih poruka, svako upoređuje verziju koju je poslao sa verzijom koju je primio, kako bi se odredio standard koji će se koristiti. Ukoliko se pregovori oko verzije završe uspešno, komunikacija se nastavlja. U suprotnom, šalje se poruka **OFPT_ERROR** i konekcija se zatvara.

4.3. *Prekid konekcije*

U situacijama ako svič izgubi konekciju ka kontroleru, mora pokušati da kontaktira neki od *backup* kontrolera. Redosled po kome svič pokušava da uspostavi komunikaciju sa *backup* kontrolerima nije specificiran samim protokolom. U slučaju gubitka svake komunikacije sa kontrolerom svič se automatski prebacuje u jedan od dva moguća režima rada: (1) *fail secure mode* ili (2) *fail standalone mode*.

U *fail secure* režimu rada, poruke i paketi koji su namenjeni kontroleru se odbacuju. U režimu *fail standalone* svič se ponaša kao *legacy* svič ili ruter. Nakon ponovne uspostave komunikacije, kontroler odlučuje da li će zadržati tabele tokova ili će ih obrisati.

Sve odluke vezano za upravljanje paketima se donose centralizovano u kontroleru, i kao takve se implementiraju u tabele tokova na samom sviču u vidu određenih pravila. Na ovaj način, kompletno procesiranje paketa se svodi na brzo prosleđivanje istih, za razliku od klasičnog pristupa gde se troši više vremena jer je potrebna određena obrada paketa na višem nivou (rutiranje, filtriranje itd.).



Slika 3. *Primer tabele toka*

OpenFlow kontroleri rade u jednom od dva definisana režima rada – reaktivno i proaktivno. U reaktivnom režimu rada, *flow* pravila se instaliraju na mrežne uređaje tek nakon inspekcije prvog paketa u toku. Ovakav pristup štedi memoriju, ali dodaje određeno vreme prilikom instaliranja pravila.

Proaktivni pristup podrazumeva popunjavanje *flow* tabela mrežnih uređaja unapred, što podrazumeva pažljivo upravljanje i planiranje kako bi se pokrile sve moguće rute paketa.

Postoji nekoliko raspoloživih platformi za razvoj OpenFlow kontrolera od kojih se izdvajaju POX i Ryu kontroleri bazirani na Python-u, Trema bazirana na C/Ruby platformi i Beacon, Floodlight i OpenDaylight platforme bazirane na programskoj jeziku Java.

5. MININET

S obzirom da su softverski definisane mreže relativno nov koncept, postoji određeni strah kod mrežnih administratora kada je u pitanju testiranje mogućnosti koje ova paradigma donosi [9]. Pre svega, teško da je neko spreman da nove protokole ili tehnologije testira u okviru postojeće produkcione mreže, a u situacijama kada je to i moguće, nije lako izolovati stvarni saobraćaj od eksperimentalnog. Upravo iz pomenutih razloga, pre puštanja određenih servisa i protokola u produkciju, poželjno je iste testirati u nekom kontrolisanom okruženju. Jedno od takvih okruženja je Mininet, koje je opisano u ovom radu.

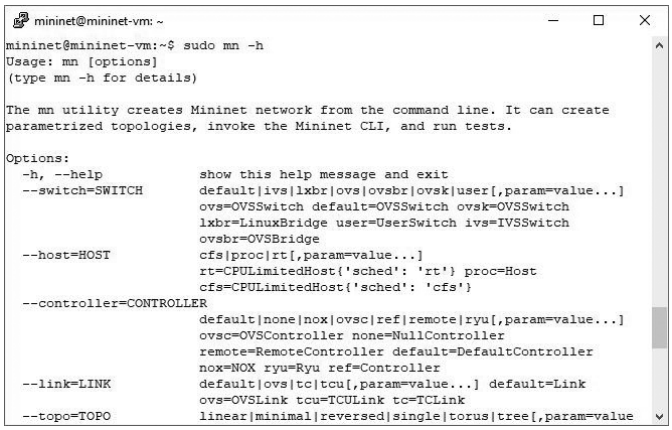
Mininet predstavlja simulaciono okruženje za kreiranje i testiranje virtuelnih mreža. Mininet, kao okruženje, se može instalirati na lokalnom računaru, u oblaku ili pak unutar virtuelne mašine. U ovom radu korišćena je već pripremljena VM koja je preuzeta sa zvaničnog sajta projekta. U pitanju je verzija Mininet 2.2.2 na Ubuntu 14.04 LTS serveru [10].

Pored pomenutog softvera, kao osnova za virtuelizaciju korišćen je Virtualbox 5.1.22, i Xming server koji služi za pokretanje X11 aplikacija kroz SSH konekciju (Wireshark, Xterm i sl.).

Nakon podizanja virtuelne mašine i provere njene IP adrese, moguće je izvršiti *remote* konekciju na pomenutu VM upotrebom alata Putty. Prilikom podešavanja parametara za SSH pristup obavezno je uključiti *X11 forwarding* opciju.

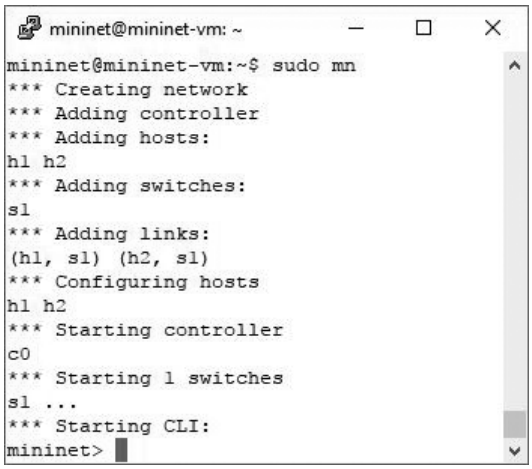
Unutar samog okruženja se može simulirati kolekcija krajnjih računara (*host*), mrežnih uređaja (*switch* ili *router*), kontrolera i linkova. Svi pomenuti entiteti se u okviru okruženja pokreću kao zasebni procesi na Linux serveru.

Nakon pristupa Mininet okruženju moguće je pregledati spisak svih mogućih opcija upotrebom komande **$ sudo mn -h**.



Slika 4. *Pregled opcija koje pruža Mininet*

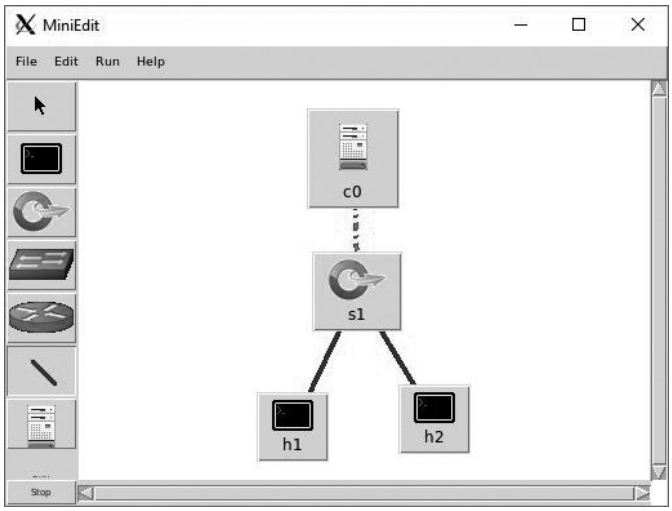
Minimalna početna topologija se kreira komandom **$ sudo mn**.



Slika 5. *Kreiranje početne topologije*

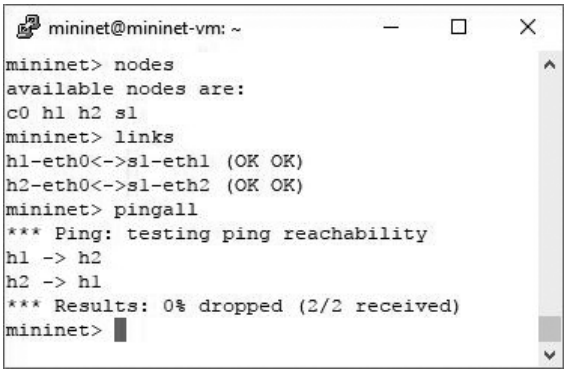
Ovom prilikom kreiran je jedan OpenFlow svič na koji su povezana dva hosta, kao i OpenFlow kontroler. Drugi način za dobijanje istog rezultata je upotreba komande **$ sudo mn --topo minimal**. Upotrebom pomenutih komandi kontroler koji se kreira u topologiji se nalazi u samoj VM, što ne mora uvek da bude slučaj. Moguće je povezati eksterni kontroler sa Mininet okruženjem, što je i prikazano u ovom radu na sledećem primeru.

Za kreiranje topologije ne mora se koristiti isključivo konzola, moguće je isti efekat postići upotrebom grafičkog alata koje je napisan u Python-u, a poziva se preko komande **$ sudo mininet/examples/miniedit.py**.



Slika 6. *Miniedit alat*

Nakon kreiranja topologije moguće je proveriti koji su čvorovi kreirani, na koji način su povezani, i da li su međusobno vidljivi tj. da li postoji konektivnost među njima. Komande kojima se vrši navedena provera su, respektivno, **nodes**, **links**, **pingall**. Rezultat izvršavanja je prikazan na slici ispod.



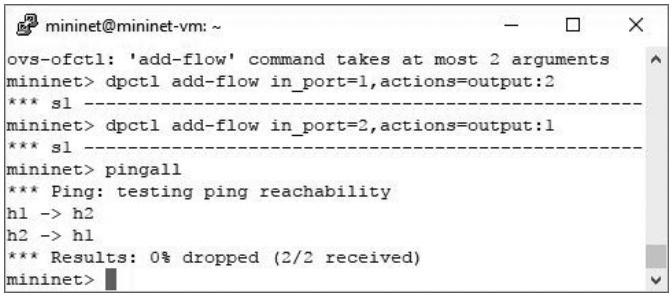
Slika 7. *Provera kreirane topologije*

Kao što je već u ovom radu pomenuto, osnovna ideja SDN koncepta je da se mrežom upravlja centralizovano, iz jedne tačke, putem kontrolera. Kontroler upravlja tabelama tokova i programira ponašanje mrežnih uređaja. Upravo iz tog razloga sledeći primer prikazuje topologiju u kojoj je kreiran eksterni kontroler, ali nije pokrenut, pa je potrebno definisati pravila u *flow* tabeli kako bi se uređaji međusobno videli.

Topologija je kreirana komandom **$ sudo mn --topo single,2 --controller remote**. Nakon provere topologije, kao u prethodnom primeru, može se videti da ne postoji konektivnost između uređaja.

Slika 8. *Provera kreirane topologije*

Razlog je nepostojanje pravila u tabelama tokova na svičevima. Ručni unos pravila se vrši komandom **dpctl add-flow** i prikazan je na slici ispod. Takođe, vidi se da je nakon dodavanja pravila uspostavljena komunikacija između hostova.



Slika 9. *Ručno dodavanje pravila toka*

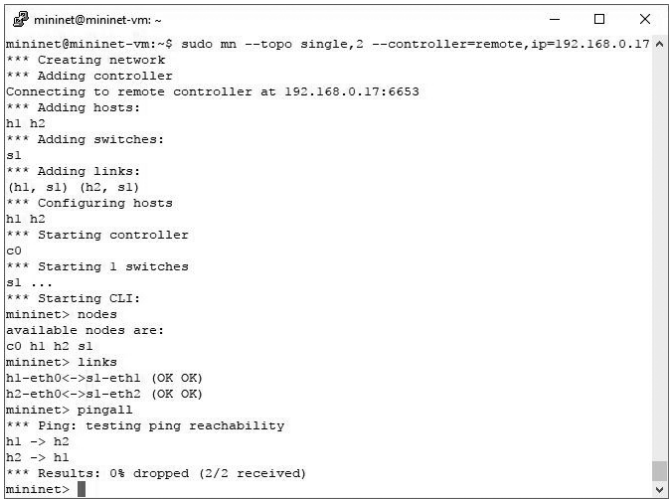
5.1. *Opendaylight*

Opendaylight je projekat otvorenog koda pokrenut 2013. godine od strane *The Linux Foundation*, sa ciljem da se promoviše softverski definisane mreže i virtuelizaciju mrežnih funkcija (NFV). Opendaylight je kontroler napisan u Javi i podržava OpenFlow protokol [11].

U nastavku je prikazana implementacija ovog kontrolera na postojeću topologiju kreiranu u Mininet okruženju.

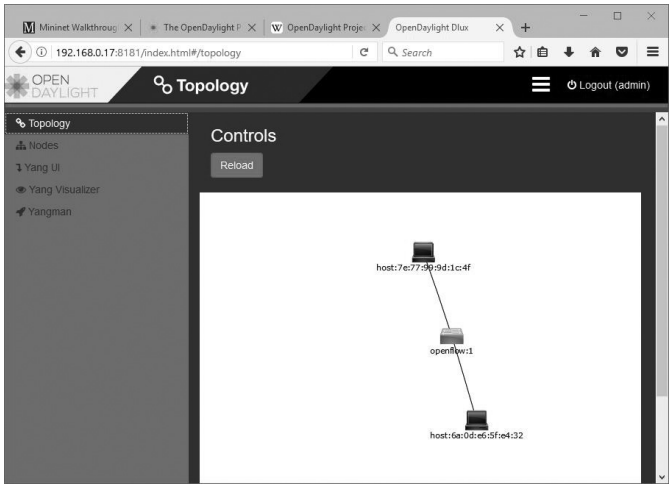
Verzija koja je korišćena je Carbon, a instalacija je odrađena u VM pod Ubuntu 14.04 LTS operativnim sistemom na kome je prethodno instalirana Java 1.8. Nakon pokretanja kontrolera potrebno je u Mininetu definisati novu topologiju i eksplicitno navesti IP adresu na kojoj se nalazi eksterni kontroler.

Slika 10. *Pokrenut ODL kontroler*



Slika 11. *Kreiranje topologije i povezivanje sa ODL kontrolerom*

Nakon navedenih koraka potrebno je u browseru uneti adresu na kojoj se nalazi kontroler u formatu **http://192.168.0.17:8181/index.html**, nakon čega se, ako je sve u redu, otvara ODL *web* interfejs. Nakon logovanja korisniku je dostupan grafički korisnički interfejs za upravljanje kontrolerom i celokupnom mrežom.



Slika 12. *Web GUI ODL kontrolera*

Na slici iznad može se videti grafički prikaz topologije koja je kreirana u Mininet okruženju.

6. PROBLEM BEZBEDNOSTI

Poseban i veoma značajan aspekt softverski definisanih mreža je bezbednost. SDN poboljšavaju mrežnu sigurnost samim tim što se ponašanjem mreže upravlja centralizovano. Lakše se uvode bezbednosne polise i smanjuje se rizik od mogućih kolizija istih. Aplikacije za monitoring saobraćaja mogu da zahtevaju od kontrolera da im povremeno prosleđuje uzorke paketa. Nakon analize moguće je određeni saobraćaj blokirati ili preusmeriti.

Prilikom promene ili ažuriranja sigurnosnih polisa potrebno je izvršiti ažuriranje samih aplikacija ili implementirati sigurnosne module na kontrolnom sloju, što je svakako jednostavnija procedura od zamene samog hardvera ili ažuriranja firmvera na istom, kao što je to princip u konvencionalnim računarskim mrežama [12].

Odvajanjem ravni upravljanja i prosleđivanja javljaju se novi sigurnosni izazovi. Upravljačka ravan postaje najzanimljiviji deo arhitekture za napad. SDN kontroler postaje usko grlo (*single point of failure*) jer u slučaju kompromitovanja može doći do pada cele mreže. Sasvim je sigurno da će lista sigurnosnih izazova biti sve veća kako SDN paradigma bude imala veću primenu u praksi.

7. ZAKLJUČAK I DALJI RAD

U situaciji kada se potrebe korisnika konstantno menjaju, a struktura saobraćaja usložnjava, softverski definisane mreže mogu biti rešenje za fleksibilno upravljanje mrežom. U radu je opisana arhitektura softverski definisanih mreža i OpenFlow protokol koji se koristi za komunikaciju između kontrolnog i sloja infrastrukture. Ukazano je i na potencijalne probleme i bezbednosne rizike koje ova paradigma nosi sa sobom. Takođe, predstavljen je i postupak rada u simulacionom okruženju Mininet, kao i povezivanje sa eksternim kontrolerom Opendaylight.

U daljem radu fokus će biti na boljem upoznavanju sa Opendaylight kontrolerom, njegovim mogućnostima i načinima upravljanja mrežnim uređajima. Takođe, u planu je i dalje istraživanje i testiranje mogućnosti Mininet okruženja, kao i pronalaženje alternativnih simulatora, i testiranje postojećih *cloud* rešenja kao što su Geni, Emulab, Deterlab i dr.

Bez obzira na poteškoće, pre svega neinteroperabilnosti uređaja i izostanka podrške za otvorene standarde od strane proizvođača hardvera, SDN paradigma ima potencijal da u potpunosti zameni tradicionalni pristup u dizajniranju i upravljanju računarskim mrežama.

REFERENCE

[1] T. Matejić, Đ. Mihailović, "Primena cloud computing okruženja u razvoj platforme za elektronsko učenje", InfoM, No. 50/2014
[2] B. Marčeta, M. Minović, M. Milovanović "Softverski definisane mreže – OpenFlow protokol", Infotech 2017
[3] https://www.sdxcentral.com/sdn/definitions/inside-sdn-architecture/
[4] https://www.opennetworking.org/sdn-resources/sdn-definition
[5] N. McKeown, T. Anderson, H. Balakrishnan, J. Rexford, "OpenFlow: Enabling innovations in campus networks", 2008
[6] https://openfine.wordpress.com/2015/02/06/learning-about-southbound-and-northbound-in-sdn-openfine-com/
[7] https://www.geni.net/
[8] "OpenFlow Switch Specification", Version 1.1.0 Implemented, 2011
[9] Marko J. Mišić, Slavko R. Gajin, "Korišćenje Mininet okruženja za simulaciju softverski definisanih mreža", TELFOR 2014
[10] http://mininet.org/walkthrough/
[11] https://www.opendaylight.org/
[12] M. Fernandez, "Evaluating OpenFlow Controller Paradigms", ICN 2013

Bojan Marčeta, Saradnik u nastavi. Univerzitet u Beogradu, Fakultet organizacionih nauka
Kontakt: bojan.marceta@mmklab.org
Oblasti interesovanja: Računarske mreže, Linux administracija

Milođ Živadinović, Student master studija, Univerzitet u Beogradu, Fakultet organizacionih nauka
Kontakt: mzdv@outlook.com
Oblasti interesovanja: Računarska bezbednost, kriptovalute