

Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu Tehnička dokumentacija Verzija 2.0

Studentski tim: Borna Ivanković
Dinko Gregorić
Filip Štimac
Vedran Serenčeš

Nastavnik: Doc. dr. sc. Ognjen Dobrijević

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Sadržaj

1. Opis projektnog zadatka	5
2. Programski upravljane komunikacijske mreže	7
2.1 Arhitektura	8
2.2 Specifikacija OpenFlow	10
2.2.1 OpenFlow komutator	11
2.2.2 OpenFlow poruke	13
3. Upravljački uređaj OpenDaylight	15
3.1 Osnovni koncept upravljačkog uređaja OpenDaylight	15
3.2 Programska struktura upravljačkog uređaja	18
3.3 Upute za instalaciju i konfiguraciju postavki upravljačkog uređaja OpenDaylight	31
3.3.1 Instalacija ovisnosti	31
3.3.2 Instalacija Maven alata	31
3.3.3 Instalacija upravljačkog uređaja	32
4. Integracija upravljačkog uređaja OpenDaylight u laboratorijsku mrežu	35
4.1 Programsko proširenje izvornog algoritma	35
4.1.1 Funkcijski zahtjevi	36
4.1.2 Dijagrami razreda	36
4.1.3 Programsko proširenje	38
4.2 Algoritam usmjeravanja zasnovan na optimizaciji kolonijom mrava	39
4.2.1 Osnovni koncept algoritma	40
4.2.2 Integracija algoritma u upravljački uređaj OpenDaylight	41
5. HTTP aplikacijsko sučelje za upravljanje kvalitetom višemedijskih usluga	43
5.1 Arhitektura klijent-poslužitelj	43
5.2 Osnovni koncept protokola HTTP	45
5.3 Model HTTP API-ja za upravljanje kvalitetom višemedijskih usluga	46
5.4 Programska izvedba HTTP API-ja	47
5.4.1 Funkcijski zahtjevi	47
5.4.2 Struktura programske izvedbe HTTP API-ja izrađenog korištenjem nestandardnih Java biblioteka	49
5.4.3 Dijagram razreda s opisom	49
5.4.4 Dijagram objekata s opisom	53
5.4.5 Struktura programske izvedbe HTTP API-ja izrađenog korištenjem standardnih Java biblioteka	54
5.4.6 Primjer izvođenja API-ja	57
6. Demonstracija rada upravljačkog uređaja OpenDaylight	59
7. Zaključak	63

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.
8. Literatura	64
9. Skraćenice	66
10. Rječnik pojmova	68
Popis slika	69
Dodatak A - Programska izvedba HTTP API-ja	71
Dodatak B - Programska izvedba funkcionalnih proširenja izvornog koda ACO zasnovanog algoritma	87

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Dnevnik promjene dokumentacije

Rev.	Opis promjena	Autor(i)	Datum
0.1	Dodan opis projektnog zadatka	Borna Ivanković	22.12.2015.
0.2	Proširen opis projektnog zadatka	Vedran Serenčes	27.12.2015.
0.3	Dodan rječnik pojmova	Vedran Serenčes	27.12.2015.
0.4	Dodan popis literature	Vedran Serenčes	27.12.2015.
0.5	Dodano poglavlje 2	Vedran Serenčes	07.01.2016.
0.6	Dodano poglavlje 5	Dinko Gregorić Borna Ivanković	10.1.2016.
0.7	Dodano poglavlje 3	Filip Štimac	11.01.2016.
0.75	Popravljen format dokumenta Dodane slike	Vedran Serenčes	11.01.2016.
0.8	Dodano poglavlje 4	Vedran Serenčes	11.01.2016.
0.9	Dodane upute za instalaciju	Borna Ivanković Vedran Serenčes	11.01.2016.
0.9.2	Formatiran izgled poglavlja 5	Dinko Gregorić	13.01.2016.
0.9.3	Izmjena strukture dokumenta	Vedran Serenčes	14.01.2016.
1.0	Dodana poglavlja 5.4.5, 5.4.6	Vedran Serenčes	21.01.2016.
1.1	Dodan zaključak	Dinko Gregorić	21.01.2016.
1.2	Dodane skraćenice	Dinko Gregorić	21.01.2016.
1.3	Dodan dodatak A	Vedran Serenčes	21.01.2016.
1.4	Dodan dodatak B	Filip Štimac Vedran Serenčes	22.01.2016.
1.6	Dodano poglavlje 6	Filip Štimac Vedran Serenčes	22.01.2016.
1.6.1	Proširen zaključak	Borna Ivanković	22.01.2016.
1.6.2	Formatiran izgled, ispravljene pogreške	Vedran Serenčes Dinko Gregorić	23.01.2016.
2.0	Konačna verzija tehničke dokumentacije	Vedran Serenčes	24.01.2016.

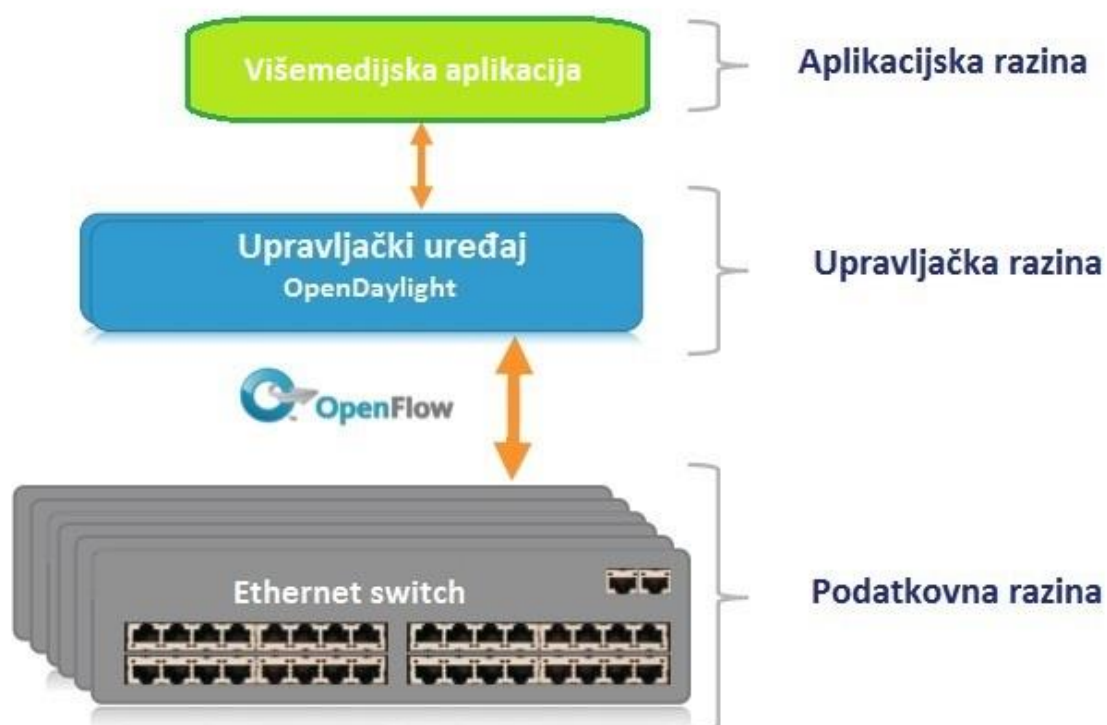
Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

1. Opis projektnog zadatka

Internet, najveća računalna komunikacijska mreža, osim po svojoj velikoj rasprostranjenosti i eksponencijalno rastućoj popularnosti, poznata je i po svojoj tromosti razvoja i spore primjene novih tehnologija. Te činjenice su posebno vidljive na primjeru IP standarda, gdje je, primjerice, čak i nakon 20 godina od početka rada na verziji IPv6 protokol-a potpuna integracija u internetsku mrežu još uvijek nije završena. Kao jedno rješenje za navedene probleme ponuđen je SDN (engl. Software-Defined Networking), odnosno programski upravljana komunikacijska mreža.

U programski upravljanoj komunikacijskoj mreži arhitektura je podijeljena na kontrolnu razinu, koju predstavlja središnji upravljački uređaj, i podatkovnu razinu podijeljenu između mrežnih uređaja. U ovakvom komunikacijskom sustavu mrežni uređaji obavljaju samo funkciju prosljeđivanja tokova podataka, dok se funkcija njihovog usmjeravanja izvodi na logički centraliziranom upravljačkom uređaju. Definiciju načina komunikacije između upravljačkog uređaja i ostalih mrežnih uređaja sadrži OpenFlow specifikacija. Središnji upravljački uređaj korišten unutar ovoga projekta realiziran je u obliku otvorene programske platforme pod nazivom OpenDaylight. Zadatak ovog projekta jest proučiti i opisati koncept programski upravljane mreže i specifikacije OpenFlow s naglaskom na različite distribucije upravljačkog uređaja OpenDaylight te njegovu integraciju u laboratorijsku mrežu na Zavodu za telekomunikacije. Potrebno je proučiti koncept i programsko rješenje algoritma usmjeravanja zasnovanog na optimizaciji kolonijom mrava, koji omogućava odabir optimalnih puteva zasnovan na vrsti toka podataka i parametrima kvalitete višemedijske usluge, te oblikovati i implementirati njegova proširenja potrebna za rad u laboratorijskoj mreži. Također, za realizaciju navedenih zadataka potrebno je oblikovati i implementirati HTTP aplikacijsko sučelje koje će služiti kao posrednik u prijenosu parametara višemedijskih usluga između klijenata (višemedijskih aplikacija) i upravljačkog uređaja OpenDaylight. Krajnji cilj projekta bit će omogućavanje povezivanje višemedijskih aplikacija i središnjeg upravljačkog uređaja (*Slika 1-1*) koristeći jedan od izvedenih HTTP API-ja, s ciljem pronalaska optimalnog puta u mreži za zadanu višemedijsku uslugu putem gore navedenog algoritma.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Slika 1-1 Studijski slučaj: integracija višemedijskih aplikacija i uređaja OpenDaylight

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

2. Programski upravljane komunikacijske mreže

Postojeća komunikacijska mrežna infrastruktura uglavnom može odgovoriti na promjenjive zahtjeve upravljanja prometnim tokovima te pružanje diferencirane kvalitete usluge (engl. *Quality of Service*, QoS) i sigurnosnih razina za pojedine tokove, ali cjelokupni proces može biti iznimno dugotrajan i zahtjevan, posebno ako je mrežna infrastruktura velika i/ili se sastoji od mrežnih uređaja različitih proizvođača. U tom slučaju, mrežni administratori moraju odvojeno konfigurirati uređaje pojedinih proizvođača i prilagoditi sve potrebne sigurnosne i druge parametre što može trajati satima ili čak danima [6]. Osim toga, povećan broj korisnika mreže, računarstvo u oblaku (engl. *cloud computing*), sigurnost, prikaz informacija u stvarnom vremenu i niz ostalih zahtjeva čini današnju mrežu vrlo „ranjivom“ u smislu performansi. Arhitekturni model tradicionalnih mreža temeljenih na prethodno programiranim uređajima čije karakteristike ovise od proizvođača do proizvođača gotovo onemogućuju provedbu ispitivanja i u konačnici implementacije novih komunikacijskih protokola. Upravo navedeni, ali i još mnogi drugi razlozi, doveli su do potrebe mijenjanja današnjeg modela komunikacijske mreže na model koji će omogućiti lakše dodavanje novih funkcionalnosti i upravljanje postojećim mrežama.

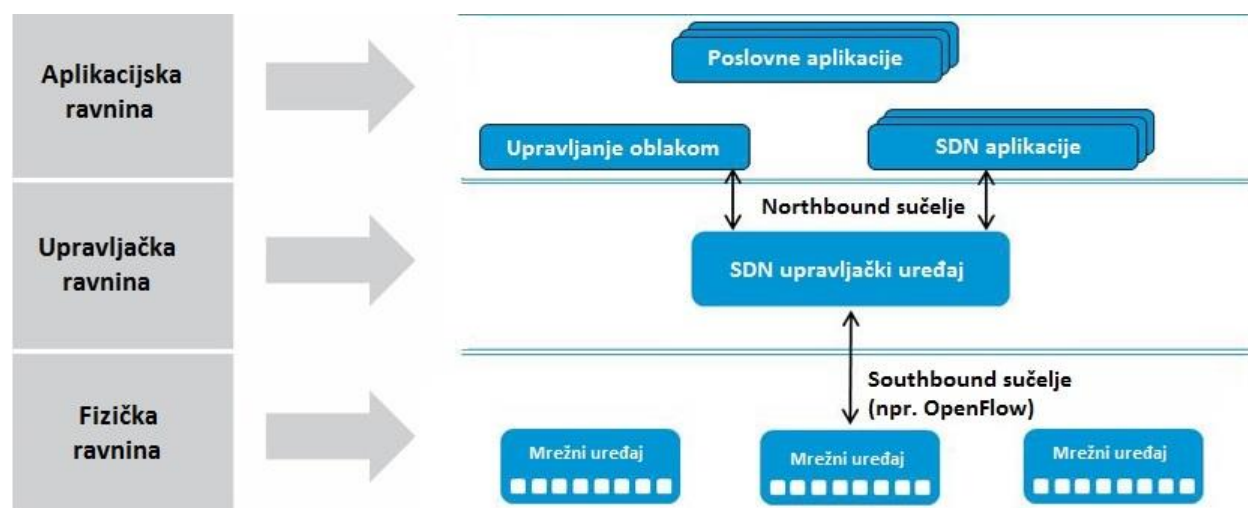
Kao jedno od rješenja koje se nameće jesu programski upravljane komunikacijske mreže (engl. *Software-Defined Networking*, SDN) [5] implementirane, primjerice, kroz OpenFlow specifikaciju [4]. Programski upravljane mreže predstavljaju moderni pristup umrežavanja koji otklanja kompleksnu i statičnu prirodu tradicionalnih mrežnih izvedbi uporabom standardne programske apstrakcije između upravljačke i postojeće podatkovne ravnine. U prevedenom značenju, osnovna ideja SDN-a je razdvajanje upravljanja mrežom od prosljeđivanja podataka koji prolaze kroz mrežu. U usporedbi s tradicionalnim komunikacijskim mrežama koje su integrirane vertikalno na postojećoj infrastrukturi, otvorena priroda SDN mreža predstavlja temeljni pomak u mogućnosti izbora, agilnosti, automatizacije i cijene troškova pojedinih mrežnih operacija koje prije nisu bile moguće. Iako se sa programski upravljanim mrežama i OpenFlow standardom započelo kao akademskim eksperimentom, oni su tijekom zadnjih nekoliko godina stekli značajan utjecaj u industriji [3]. Danas većina proizvođača mrežnih uređaja uključuje podršku OpenFlow API-ja u svoju opremu. Dokaz da je SDN zamah bio dovoljno snažan predstavlja osnivanje Open Network Foundation (ONF) organizacije, od strane

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

tvrtki kao što su Google, Facebook, Yahoo, Microsoft, Verizon i Deutsche Telekom, koja se brine za razvoj, promociju i usvajanje koncepta programski upravljanih mreža.

2.1 Arhitektura

Mrežna arhitektura SDN modela (*Slika 2-1*) sastoji se od tri temeljna sloja (ravnine): fizički (engl. *infrastructure plane*), upravljački (engl. *control layer*) i aplikacijski (engl. *application layer*). Svaka ravnina sadrži specifične funkcije i karakteristike vezane isključivo uz tu ravninu.



Slika 2-1 Arhitektura programski upravljane komunikacijske mreže

Fizička ravnina, odnosno infrastrukturni sloj SDN arhitekture sličan je tradicionalnoj arhitekturi komunikacijskih mreža i sastoji se od mrežnih uređaja kao što su komutatori, usmjeritelji i posrednički uređaji (engl. *middlebox appliances*). Glavna razlika leži u činjenici da ti fizički uređaji sadrže samo jednostavnu funkciju prosljeđivanja paketa, bez ugrađene upravljačke logike ili programa koji donosi autonomne odluke. Također, odluke preusmjeravanja paketa transformirane su iz odredišno orijentiranih u odluke ovisne o tokovima, kao skupovima paketa s odgovarajućim pratećim vrijednosnim kriterijima i zadanim funkcijama. Na taj način, paketi istog toka se jednako interpretiraju i procesiraju u svim uređajima fizičkog sloja, čime se postiže apstrakcija tokova kojom se omogućuje generalizacija rada različitih mrežnih uređaja[5].

Integracija upravljačkog uređaja OpenDaylight u programski upravljani laboratorijску mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Iznad fizičke ravnine nalazi se upravljački sloj kojeg predstavlja tzv. mrežni operacijski sustav (engl. *Network Operating System*, NOS) ili SDN upravljački sustav koji se sastoji od jednog ili više upravljačkih uređaja. Takvi logički centralizirani SDN upravljači sadrže svu mrežnu inteligenciju i održavaju globalni pogled na mrežu zbog čega često nose naziv „mozak“ mreže. Drugim riječima, središnji upravljački uređaj obavlja sve zahtjevne operacije uključujući usmjeravanje, imenovanje, dodjeljivanje pravila pojedinim tokovima kao i sigurnosne operacije. Upravljački uređaj oblikovan je kao programska platforma koja može biti pokrenuta na prikladnim fizičkim uređajima (osobna računala, poslužitelj i sl.) ili na nekom virtualnom računalu. U sklopu ovog projekta koristit će se SDN upravljački uređaj OpenDaylight Hydrogen pokrenut na jednom od laboratorijskih računala. Komunikacija i upravljanje upravljačkog uređaja podatkovnom razinom odvija se putem OpenFlow protokola preko standardnih grupa poruka, koje su detaljnije opisane u poglavlju 2.2.2. Na taj način mreža prema aplikacijskom sloju djeluje kao jedan logički komutator, gdje aplikacije stječu nadzor nad cijelom mrežom s jednog logičkog mjesta, neovisno o proizvođaču mrežne opreme [5][6].

Povrh upravljačkog sloja na vrhu cjelokupne SDN arhitekture nalazi se aplikacijski sloj koji obuhvaća poslovne i mrežne aplikacije. Pod pojmom aplikacije podrazumijeva se usluga koju mrežni operator pruža korisniku. Karakteristike SDN mreže omogućuju da aplikacije određuju resurse i način ponašanja mreže, odnosno, mreža je programibilna kroz programske aplikacije koje se nalaze na vrhu NOS-a, što predstavlja temeljnu karakteristiku SDN-a. Sučelje između upravljačke i aplikacijske razine naziva se aplikacijsko-upravljačko sučelje (engl. *application plane interface*, A-CPI) poznatije pod imenom *northbound* sučelje (engl. *northbound interface*, NBI). Također, SDN aplikacije mogu pozivati i druge vanjske usluge kao i upravljati bilo kojim brojem SDN upravljačkih uređaja pri obavljanju svojih zadataka. Unutar ovoga projekta, SDN aplikacije predstavljaju moduli unutar upravljačkog uređaja koji računaju statistiku mreže i optimalni put, kao i klijentska aplikacija koja upravljačkom uređaju šalje parametre višemedijskih usluga[5].

Ključan dio arhitekture programski upravljanih mreža predstavljaju sučelja između njezinih ravnina. Putem njih se vrši komunikacija između pojedinih razina. Dva ključna sučelja koja omogućavaju komunikaciju između fizičke i upravljačke razine, odnosno upravljačke i aplikacijske razine nazivaju se *southbound* i *northbound* sučelja.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Northbound sučelja u SDN arhitekturi omogućuju komunikaciju između elemenata upravljačke i aplikacijske razine SDN mreže. Izvedena su u obliku programskog okruženja te vrše apstrakciju skupa instrukcija niske razine korištene od strane *southbound* sučelja pri programiranju jednostavnih mrežnih uređaja. Također, ne postoji općenito prihvaćeni standard ovih sučelja i trenutno se njihova implementacija temelji na *ad-hoc* principu, tj. prema potrebama pojedinih aplikacija. Iz tog razloga je postojanje otvorenog i standardnog oblika *northbound* sučelja ključno za mobilnost i interoperabilnost aplikacija korištenih širom različitih upravljačkih platformi [1][5].

Southbound sučelja predstavljaju „mostove“ koji povezuju elemente upravljačke i fizičke razine. Osnovna zadaća *southbound* sučelja jest omogućiti SDN upravljačkom uređaju programiranje, odnosno komunikaciju s ostalim mrežnim uređajima prema određenim zahtjevima, među kojima su i oni dobiveni od strane mrežnih aplikacija[1][5]. U našem slučaju to je pronalazak optimalnog puta u mreži na temelju dobivenih parametara višemedijskih usluga. Upravo OpenFlow specifikacije predstavljaju najrasprostranjeniji i najviše prihvaćen primjer *southbound* sučelja.

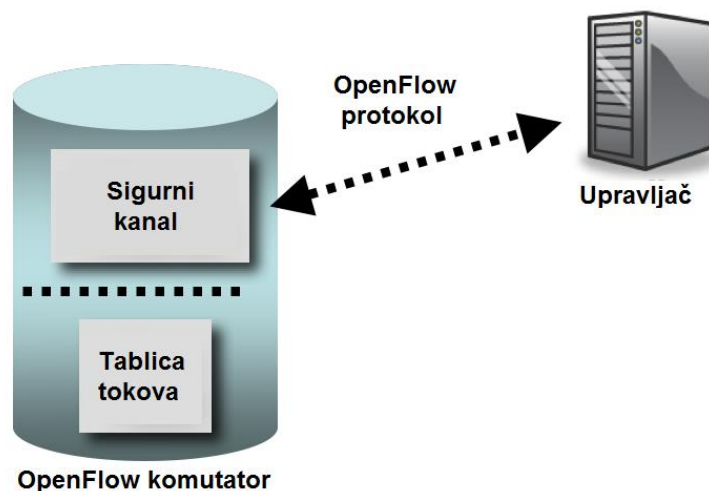
2.2 Specifikacija OpenFlow

OpenFlow specifikacija [4] predstavlja skup pravila koja omogućuju komunikaciju upravljačkog i fizičkog sloja. Sastoji se od protokola za komunikaciju te definiciju komponenti i skupa funkcionalnosti potrebnih za ispravno funkcioniranje svakog mrežnog uređaja. Komunikacija OpenFlow protokolom između upravljačkog uređaja i komutatora odvija se preko sigurnog kanala (*Slika 2-2*) putem TLS (engl. *Transport Layer Security*) ili SSL (engl. *Secure Sockets Layer*) protokola.

OpenFlow pruža otvoreni protokol za programiranje tablica tokova u različitim komutatorima i usmjeriteljima. Mrežni administrator može podijeliti promet u produkcijske i istraživačke tokove. Tako znanstvenici mogu kontrolirati vlastite tokove odabirom puta kojim će paketi putovati te njihovim načinom obrade. Na taj način olakšava se testiranje novih protokola za usmjeravanje, sigurnosnih modela, shema za adresiranje pa čak i alternativa IP protokolu. U isto vrijeme, produkcijski promet je odvojen i obrađuje se na isti način kao i danas[3][4].

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

OpenFlow protokol nastao je na sveučilištu Stanford, a njegovu standardizaciju preuzela je ONF organizacija. Trenutno postoji šest verzija OpenFlow specifikacije koje se koriste kod izvedbi mrežnih uređaja, a to su 1.0, 1.1, 1.2, 1.3, 1.4 te 1.5. Svaka specifikacija opisuje opći koncept, mogućnosti primjene te probleme i njihova rješenja koja se mogu pojaviti prilikom komunikacije. U sklopu ovoga projekta korištena je verzija 1.0 OpenFlow protokola zbog njegove podržanosti od strane mrežnih uređaja unutar laboratorija, kao i samog SDN upravljačkog uređaja.



Slika 2-2 Komunikacija komutatora i upravljačkog uređaja putem OpenFlow protokola

2.2.1 OpenFlow komutator

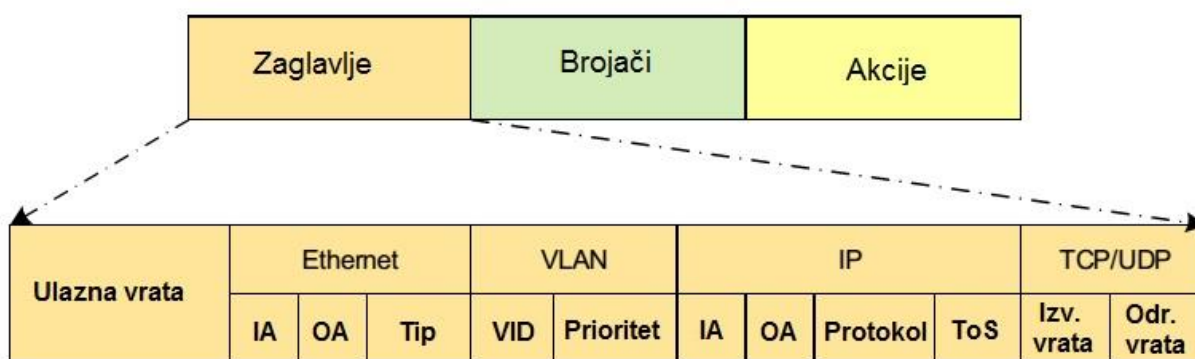
Implementacija OpenFlow protokola zahtijeva zadovoljavanje minimalno dva uvjeta: postojanje OpenFlow upravljačkog uređaja (engl. *OpenFlow controller*) i komutatora koji podržava protokol OpenFlow (engl. *OpenFlow-enabled switch*)[4].

Kod inačice 1.0 OpenFlow protokola, komutatori se sastoje od dva osnovna dijela. Prvi predstavlja hardverski dio OpenFlow komutatora i čini ga tablica tokova (engl. *flow table*), dok drugi dio predstavlja programski dio komutatora kojeg čini sigurnosni kanal (engl. *secure channel*) pomoću kojega se komutator povezuje s upravljačkim uređajem. Koristeći OpenFlow protokol upravljački uređaj može dodati, ažurirati i izbrisati unose tokova (engl. *flow entries*) unutar tablice[4].

Postoje dva načina unosa tokova unutar tablice. To su reaktivan i proaktivan način. Kod

Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

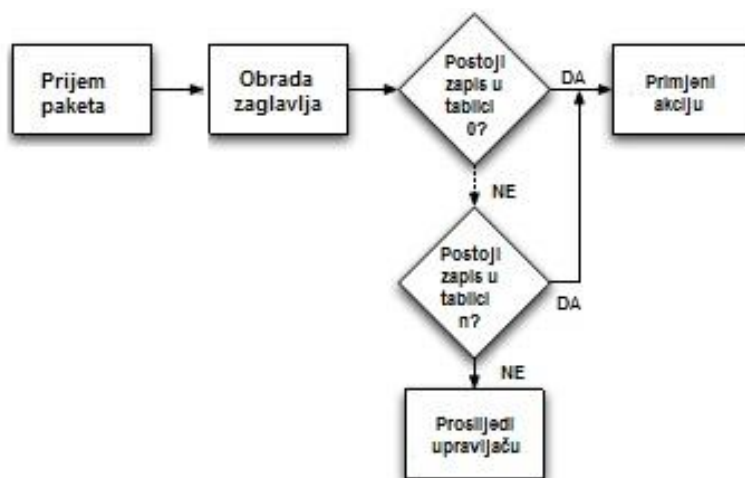
reaktivnog načina prilikom dolaska okvira koji ne prolazi tablicu tokova, on se proslijeđuje upravljačkom uređaju koji određuje daljnju akciju, odnosno hoće li dodati novi zapis u tablicu ili će okvir zajedno sa ostatkom toka biti odbačen. Na ovaj način postiže se efikasnost tablice u smislu optimalne iskoristivosti, jer ne sadrži tokove koji se vrlo malo koriste, dok u drugu ruku povećava vrijeme uspostave toka (engl. *setup time*). Kod drugog, proaktivnog načina, upravljački uređaj unaprijed popunjava tablicu tokova čime se ne dodaje dodatno vrijeme prilikom uspostave toka te se u slučaju gubitka povezanosti s upravljačem podatkovni promet ne obustavlja.



Slika 2-3 Sadržaj unosa toka (flow entry) unutar OpenFlow tablice komutatora

Kao što možemo vidjeti na *Slika 2-3* zapis tablice tokova se sastoji od 3 dijela, a to su zaglavlje, brojači i akcije. Također je prikazan i sadržaj zaglavlja tablice koji služi za uspoređivanje pristiglih okvira s ciljem prepoznavanja toka i pridjeljivanja akcija. U slučaju podudarnosti paketa s određenim tokom iniciraju se prethodno definirane akcije zapisane u polju akcija, dok se u suprotnom provode akcije sukladne konfiguraciji polja za nepostojeće tokove. Na taj način paketi koji se ne podudaraju mogu se proslijediti upravljačkom uređaju te ih odbaciti ili napraviti novi zapis unutar tablice tokova[3][4]. Navedeni scenariji procesiranja paketa u OpenFlow komutatoru prikazani su na *Slika 2-4*.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Slika 2-4 Dijagram toka obrade paketa u OpenFlow komutatoru

2.2.2 OpenFlow poruke

OpenFlow protokol podržava tri tipa poruka u komunikaciji s mrežnim uređajima. To su upravljač-komutator (engl. *controller-to-switch*), asinkrone (engl. *asynchronous*) i simetrične (engl. *symmetric*) poruke od kojih svaka sadrži podtipove.

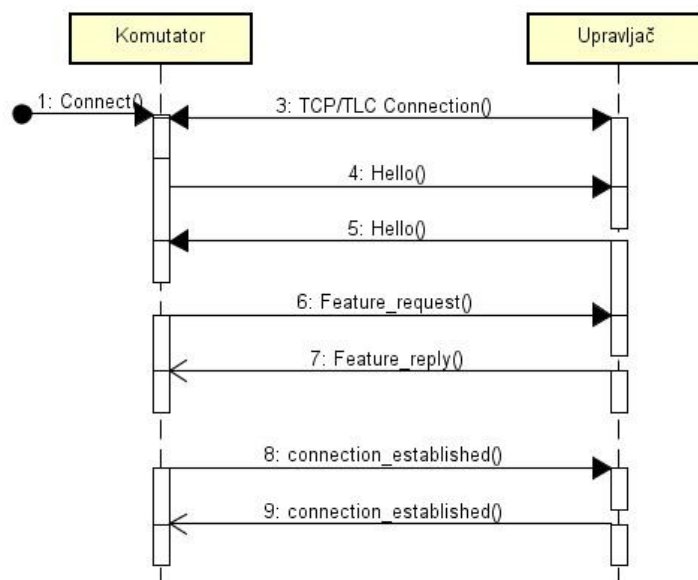
Controller-to-switch poruke inicirane su od strane upravljačkog uređaja i ne moraju zahtijevati povratnu poruku od komutatora. Ovim porukama upravljački uređaj direktno upravlja i nadzire stanje komutatora. Neke od vrsta poruka koje pripadaju ovoj skupini su poruke *Feature*, *Configuration*, *Modify-State*, *Read-State*, *Send-Packet* i *Barrier*[6].

Asinkrone poruke šalje komutator i koriste se za obavješćavanje upravljačkog uređaja o događajima u mreži i stanju samog komutatora. Postoje četiri osnovna tipa asinkronih poruka koje obavješćavaju upravljački uređaj o primljenim paketima, promjeni stanja komutatora i nastalim pogreškama u prijenosu ili obradi. Točan naziv tih poruka je, npr., *Packet-in*, *Flow-Removed*, *Port-Status* i *Error*[6].

Simetrične poruke inicirane su od strane upravljačkog uređaja ili komutatora bez prethodnog zahtjeva. Poruke koje pripadaju ovoj skupini su poruke *Hello*, *Echo* i *Vendor*. *Hello* poruke izmjenjuju se između komutatora i upravljačkog uređaja na početku uspostave veze (Slika 2-5), gdje je polje verzije OpenFlow protokola postavljeno na najveću podržanu verziju. Ukoliko se dogodi pogreška, odnosno nekompatibilnost između komutatora i upravljačkog

Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

uređaja šalje se *HelloFailed* poruka i dolazi do prekida veze.



Slika 2-5 Sekvencijski dijagram uspostave OpenFlow veze između komutatora i upravljačkog uređaja

Integracija upravljačkog uređaja OpenDaylight u programski upravljani laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

3. Upravljački uređaj OpenDaylight

3.1 Osnovni koncept upravljačkog uređaja OpenDaylight

OpenDaylight je otvorena programska platforma napisana u Javi i temeljena na *OSGi* arhitekturi [20]. Nastala je suradnjom velikog broja tehnoloških tvrtki (IBM, Microsoft, Cisco, HP, itd.) te služi ostvarivanju napretka u SDN tehnologijama. Oko platforme postoji zajednica ljudi koji rade na njoj i doprinose njezinom razvoju, a sastoji se od kodova i nacрта. Oni su temelj za ubrzanje usvajanja znanja, poticanje novih inovacija, smanjenja rizika i stvaranje transparentnijeg pristupa SDN-u.

OpenDaylight je usmjeren na izgradnju otvorene, na standardima utemeljene, SDN upravljačke platforme koja je pogodna za implementaciju u heterogenim mrežnim okruženjima. Uz modularne upravljačke radne okvire (engl. *modular controller framework*), uključuje i podršku za veliki broj standardnih i nadolazećih SDN protokola (u našem slučaju OpenFlow protokol), mrežnih usluga kao što je virtualizacija, dobro definiranih aplikacijskih sučelja i elemenata podatkovne ravnine, uključujući sučelja fizičkih uređaja i poboljšanja virtualnih komutatora.

Prva verzija OpenDaylight-a, ona koja se koristi i u ovom projektnom zadatku je verzija Hydrogen (osnovno izdanje). Ona se sastoji od 15 projekata podijeljenih u tri različita izdanja. Osnovno izdanje (engl. *Base Edition*) sastoji se od OpenDaylight upravljačkog uređaja i minimalne funkcionalnosti za njegovo testiranje. Ovo izdanje potrebno je svima koji koriste OpenDaylight. Dva proširena izdanja, virtualizacijsko izdanje (engl. *Virtualization Edition*) i izdanje za davatelje mrežnih usluga (engl. *Service Provider Edition*) uključuju projekte koji se odnose na virtualizaciju mreže i potrebe pružatelja mrežnih usluga.

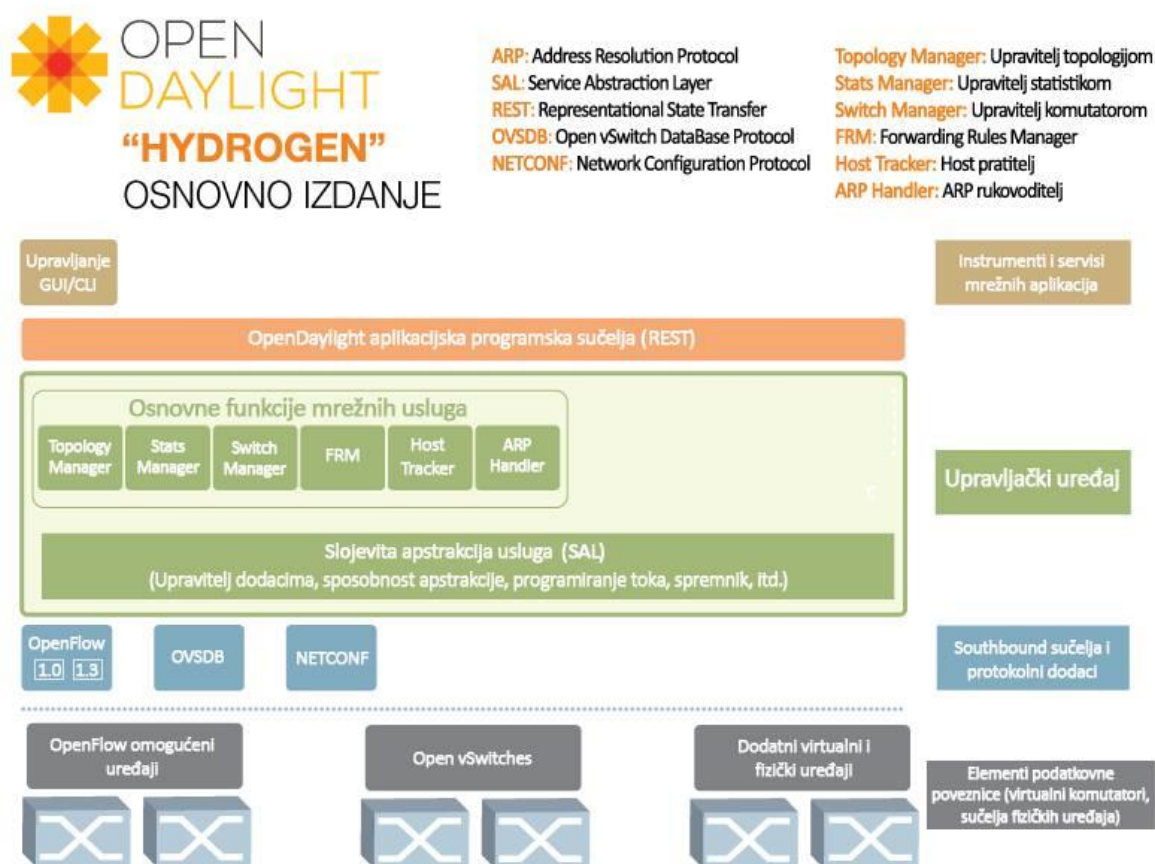
Ključne sastavnice osnovnog izdanja verzije Hydrogen[21] su:

- Upravljački uređaj: modularan, proširiv, prilagodljiv i višeprotokolni SDN upravljački uređaj temeljen na *OSGi* arhitekturi;
- OpenFlow dodatak (engl. *OpenFlow plugin*): integracija biblioteke OpenFlow protokola u upravljački SAL (engl. *Service Abstraction Layer*);

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

- Biblioteka OpenFlow protokola: implementacija biblioteka OpenFlow 1.0 i OpenFlow 1.3 protokola (opcionalno);
- OVSDb (Open vSwitch Database): konfiguracija OVSDb-a i podrška upravljačkih protokola; te
- YANG alati: mrežni konfiguracijski protokol (NETCONF) baziran na Javi, postavljanje i rad s YANG alatima na OpenDaylight projektima.

Na Slika 3-1 prikazana je arhitektura platforme OpenDaylight, verzija Hydrogen.



Slika 3-1 Arhitektura platforme OpenDaylight, verzija Hydrogen

Upravljački uređaj unutar arhitekture OpenDaylight-a u SDN mreži čini glavni upravljački dio te mreže. Sadrži osnovne funkcije mrežnih usluga i slojevit apstrakciju usluga (engl. *Service Abstraction Layer*, SAL) vođenu aplikacijskim programskim sučeljem (Slika 3-2).

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

On je strateška točka komunikacijske mreže te prenosi informacije mrežnim uređajima koristeći *southbound* API, ali i aplikacijama koristeći *northbound* API.



Slika 3-2 Shema upravljačkog uređaja unutar platforme OpenDaylight

Važna karakteristika upravljačkog uređaja koju izvršava SAL upravo je povezivanje aplikacijske ravnine s fizičkom ravninom. To se postiže na način da se aplikacije vezane uz rad na mreži, koje zahtijevaju usluge od mrežnih uređaja i protokola pomoću kojih mrežni uređaji komuniciraju, mogu izvoditi neovisno o specifikaciji uređaja na fizičkoj ravnini i protokolnih dodataka (engl. *protocol plugins*).

Djelovanje SAL-a može se opisati kao veliki registar usluga podržan raznim modulima koji spaja usluge na aplikacije koje ih zahtijevaju. Moduli koji pružaju usluge, odnosno proizvođači, mogu „registrirati“ svoje API-je unutar registra usluga. Kada aplikacija, odnosno potrošač zatraži uslugu putem generičkog API-ja, SAL je odgovoran za sastavljanje zahtjeva za uslugom povezivanjem proizvođača i potrošača u tzv. ugovor, kojim posreduje SAL.

Postoje dvije vrste slojevite apstrakcije, a to su slojevita apstrakcija usluga vođena aplikacijskim programskim sučeljem (engl. *API driven SAL*, AD-SAL) i slojevita apstrakcija usluga vođena modelom (engl. *Model driven SAL*, MD-SAL). S obzirom da koristimo verziju Hydrogen u kojoj se radi sa slojevitom apstrakcijom usluga vođenom aplikacijskim programskim sučeljem (tek u kasnijim verzijama pojavljuje se rad s MD-SAL), u nastavku teksta podrazumijeva se korištenje AD-SAL. On pruža dodatan sloj apstrakcije i omogućuje razvoj aplikacije kojoj nisu bitni protokoli na koji se SDN aplikacija oslanja. Kod takvog načina apstrakcije usluga, sva sučelja, usmjeravanja zahtjeva između pružatelja i potrošača usluge te podatkovne strukture su statički predefinirane za vrijeme izgradnje same aplikacije. To svojstvo omogućuje jednostavniji razvoj aplikacije iz pogleda pretvaranja same ideje u kôd kojim je ista implementirana, ali ima i neka ograničenja koja ne postoje kod MD-SAL. U idućem poglavlju

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

pobliže je opisana programska struktura upravljačkog uređaja i programska struktura drugog dijela upravljačke ravnine, osnovnih funkcija mrežnih usluga.

3.2 Programska struktura upravljačkog uređaja

Kao što je već navedeno, upravljački uređaj, odnosno platforma sastoji se od osnovnih funkcija mrežnih usluga i slojevite apstrakcije usluga vođene aplikacijskim programskim sučeljem. U ovome poglavlju opisani su najvažniji paketi, klase i metode, tj. programska struktura upravljačkog uređaja koja je korištena u projektu.

Korišteni paketi, izvedena sučelja i metode u implementaciji slojevite apstrakcije usluga vođene aplikacijskim programskim sučeljem navedeni su u nastavku teksta. Korišteni format za opise paketa, razreda, sučelja i metoda je sljedeći:

- Paket - naziv korištenog paketa,
- import - naziv uvezenog razreda ili sučelja,
- metoda - sastoji se od naziva i opisa metode te tipa koji predstavlja povratnu vrijednost metode.

Metode *equals* i *toString* često se pojavljuju te su, radi jednostavnosti, navedene i opisane samo prvi put gdje se pojavljuju. Opisane su samo najvažnije programske strukture. Ostale se mogu pronaći u literaturi [18].

Paket *org.opendaylight.controller.sal.core*

Import 1: **import** *org.opendaylight.controller.sal.core.Edge*

- Razred koji opisuje *Edge* (poveznicu) koji spaja dva priključka čvora (engl. *Node Connector*), poveznica (engl. *Edge*) je usmjerena te sadrži koncept glave i repa koji određuju njezin smjer.

Metoda 1: ***getHeadNodeConnector*** ()

Tip: *NodeConnector*

- Vraća glavu *NodeConnector*-a poveznice.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Metoda 2: **getTailNodeConnector** ()

Tip: *NodeConnector*

- Vraća rep *NodeConnector*-a poveznice.

Import 2: **import** *org.opendaylight.controller.sal.core.Node*

- Opisuje generički mrežni element. Čvor je opisan parom (*NodeType*, *NodeID*), *NodeType* je potreban kako bi dodatno specificirao *NodeID*.

Import 3: **import** *org.opendaylight.controller.sal.core.NodeConnector*

- Koristi se za opis generičkih mrežnih „spojnih točaka“, koje su vezane na *Node* (čvor). *Node* se identificira kao par (*NodeConnectorType*, *NodeConnectorID*).

Metoda 1: **getNode** ()

Tip: *Node*

- Vraća *Node* (čvor) tog objekta gdje je pozvana.

Metoda 2: **getNodeConnectorIdAsString** ()

Tip: *String*

- Prikaz *NodeConnector*-a kao tipa *String* bez *Node* konteksta.

Import 4: **import** *org.opendaylight.controller.sal.core.Host*

- Opisuje generički krajnji uređaj unutar mreže, npr. računalo, server, itd.

Metoda 1: **GetNetworkAddress** ()

Tip: *InetAddress*

- Vraća *NetworkAddress*.

Metoda 2: **GetNetworkAddressAsString** ()

Tip: *String*

- Vraća *NetworkAddress* kao tip *String*.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Paket *org.opendaylight.controller.sal.flowprogrammer*

Import 1: **import** *org.opendaylight.controller.sal.flowprogrammer.Flow*

- Predstavlja tok: *match* (predstavlja ispunjavanje kriterija za podudarnost paketa) + akcije/radnje za dani tok + specifična svojstva za tok.

Metoda 1: **getMatch** ()

Tip: *Match*

- Dohvati *Match* (kriterij podudarnosti) iz toka.

Metoda 2: **setIdleTimeout** (*short idleTimeout*)

Tip: *void*

Metoda 3: **setPriority** (*short priority*)

Tip: *void*

Paket *org.opendaylight.controller.sal.reader*

Import 1: **import** *org.opendaylight.controller.sal.reader.NodeConnectorStatistics*

- Prikazuje statistike za priključak čvora (engl. *Node Connector*).

Metoda 1: **getNodeConnector** ()

Tip: *NodeConnector*

- Vraća *NodeConnector*.

Metoda 2: **getReceiveDropCount** ()

Tip: *long*

- Vraća *rx* broj odbačenih paketa (engl. *drop count*) za priključak.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Metoda 3: ***getReceivePacketCount*** ()

Tip: *long*

- Vraća *rx* broj paketa za priključak.

Metoda 4: ***getTransmitDropCount*** ()

Tip: *long*

- Vraća *tx* broj odbačenih paketa (engl. *drop count*) za priključak.

Metoda 5: ***getTransmitPacketCount*** ()

Tip: *long*

- Vraća *tx* broj paketa za priključak.

Paket *org.opendaylight.controller.sal.action*

Import 1: ***import org.opendaylight.controller.sal.action.Action***

- Predstavlja generičku radnju/akciju koja je/će biti primijenjena na odgovarajući (engl. *matched*) okvir/paket/poruku.

Import 2: ***import org.opendaylight.controller.sal.action.Output***

- Predstavlja radnju/akciju slanja paketa s (fizičkih) vrata/priključka.

Import 3: ***import org.opendaylight.controller.sal.action.Controller***

- Predstavlja radnju/akciju „usmjeravanja“ (slanja po putu) do kontrolera.

Paket *org.opendaylight.controller.sal.match*

Import 1: ***import org.opendaylight.controller.sal.match.Match***

- Predstavlja generičko ispunjavanje kriterija (engl. *match criteria*) za mrežni okvir/paket/poruku.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Metoda 1: **setField** (*MatchType type, Object value*)

Tip: *void*

- Postavljač zaglavlja okvira/paketa/poruka na kojima se radi *match* (podudaranje). Za MAC adrese, treba proslijediti kloniranu vrijednost adrese u funkciju.

Paket *org.opendaylight.controller.sal.packet*

Import 1: **import** *org.opendaylight.controller.sal.packet.Packet*

- Razred koji se koristi kako bi se opisale različite protokolne jedinice podataka, poput Ethernet okvira, IP paketa i TCP segmenta.

Metoda 1: **getParent** ()

Tip: *Packet*

- Dohvati roditelja.

Metoda 1: **getPayload** ()

Tip: *Packet*

- Dohvaća korisne podatke (polje podataka u paketu).

Metoda 1: **getRawPayload** ()

Tip: *byte []*

- Vraća korisno polje podataka paketa u obliku *raw*, ako polje nije bilo parsirano. Pozivatelj može pozvati funkciju ako mu metoda *getPayload()* vraća vrijednost *null*. Vraća: polje podataka u obliku *raw* ako ga nije moguće parsirati kao polje okteta, inače vraća vrijednost *null*.

Import 2: **import** *org.opendaylight.controller.sal.packet.IListenDataPacket*

- Sučelje koje je potrebno implementirati u svim komponentama koji žele primiti OpenFlow poruku *inPacket*.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Metoda 1: ***receiveDataPacket*** (*RawPacket inPkt*)

Tip: *PacketResult*

- Rukovoditelj za primanje paketa. Aplikacija može poslati signal do SAL-a, ako je paket bio konzumiran ili ne. Vraća naznaku treba li paket još uvijek obrađivati ili njegovu obradu treba zaustaviti.

Import 3: ***import*** *org.opendaylight.controller.sal.packet.IDataPacketService <<interface>>*

- *Data Packet* usluge koje SAL pruža komponentama platforme.

Metoda 1: ***decodeDataPacket*** (*RawPacket pkt*)

Tip: *Packet*

- Dekodira paket podataka (engl. *Data Packet*) primljen kao *raw* tok podataka (engl. *raw stream*). Vraća formatiran *Data Packet*.

Metoda 1: ***encodeDataPacket*** (*Packet pkt*)

Tip: *RawPacket*

- Kodira formatiran *Data Packet* u *raw* tok okteta. Vraća *RawPacket* prikaz, dobar u svrhu prijenosa.

Metoda 1: ***transmitDataPacket*** (*RawPacket outPkt*)

Tip: *void*

- Prosljeđuje paket podataka iz čvora. Prosljeđivanje će se dogoditi jedino ako *RawPacket* ima postavljen *OutgoingNodeConnector*, inače se neće dogoditi.

Paket *org.opendaylight.controller.sal.utils*

Import 1: ***import*** *org.opendaylight.controller.sal.utils.Status*

- Predstavlja povratni objekt OSGi funkcijskih poziva uslužnih sučelja. Sadrži kod

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

StatusCode koji predstavlja rezultat poziva i *String* koji opisuje razlog kvara.

Metoda 1: ***isSuccess*** ()

Tip: *boolean*

- Pokazuje je li status uspješan. Vraća vrijednost *true* ako je statusni kod *StatusCode.SUCCESS*.

Import 2: ***import*** *org.opendaylight.controller.sal.utils.NetUtils*

- Pomoćna (engl. *utility*) klasa koja sadrži osnovne pomoćne funkcije potrebne za rad na mrežnim podatkovnim strukturama.

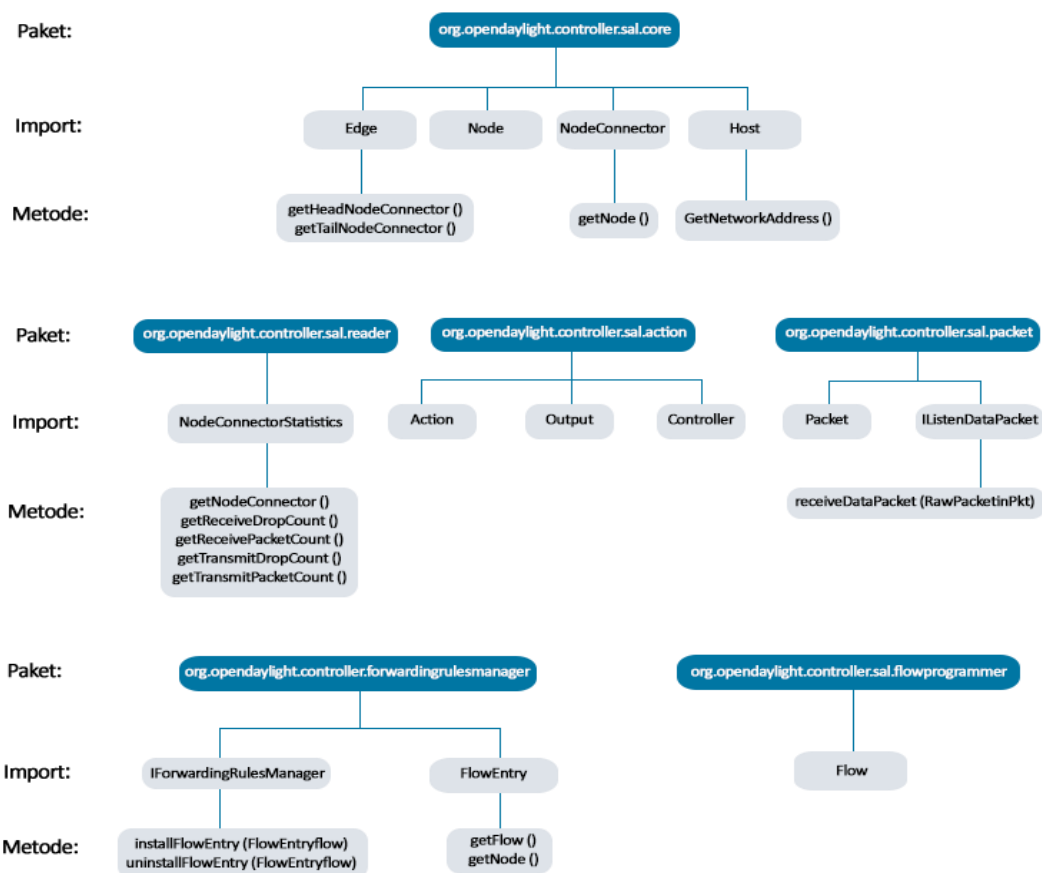
Metoda 1: ***getInetAddress*** (*int address*)

Tip: *static InetAddress*

- Pretvara IP adresu predanu kao *Integer* u odgovarajući *InetAddress* objekt. Vraća IP adresu u *InetAddress* obliku.

Na slici 3-3 nalazi se grafički prikaz najvažnijih (i najčešće upotrebljenih) paketa, razreda, sučelja i metoda.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Slika 3-3 Najvažniji korišteni paketi, razredi, sučelja i metode

Korišteni paketi, izvedena sučelja i metode u implementaciji funkcija mrežnih usluga navedeni su u nastavku teksta. Metode *equals* i *toString* često se pojavljuju te su, radi jednostavnosti, navedene i opisane samo prvi put gdje se pojavljuju.

Paket: org.opendaylight.controller.forwardingrulesmanager

Import 1: **import** org.opendaylight.controller.forwardingrulesmanager.

IForwardingRulesManager

- Sučelje koje opisuje metode za postavljanje ili uklanjanje pravila prosljeđivanja i za pristupanje bazi podataka tokova.

Metoda 1: **getFlowEntriesForNode** (Node node)

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Tip: *List<FlowEntry>*

- Vraća popis/listu svih unosa tokova (engl. *Flow*) za mrežni čvor. Ta lista sadrži tokove kakvi su bili zahtijevani za postavljanje od aplikacija, prije bilo kakvog integriranja s tablicom tokova.

Metoda 2: ***getInstalledFlowEntriesForNode*** (*Node node*)

Tip: *List<FlowEntry>*

- Vraća listu svih unosa tokova postavljenih u mrežnom čvoru. Lista sadrži djelotvorne tokove postavljene na čvorove nakon integriranja s bilo kojom dostupnom tablicom tokova. Ako niti jedna tablica tokova nije specificirana, ova metoda vraća jednaku listu kao i *getFlowEntriesForNode* (*Node node*).

Metoda 3: ***installFlowEntry*** (*FlowEntry flow*)

Tip: *Status*

- Postavlja se *FlowEntry* (unos toka). *ForwardingRulesManager* će zatražiti SDN protokolni dodatak (engl. *plugin*) da postavi tok (engl. *flow*) na mrežnom čvoru. U slučaju uspješne operacije, ažurirat će svoju bazu podataka i vratit će pripadajući statusni kod.

Metoda 4: ***uninstallFlowEntry*** (*FlowEntry flow*)

Tip: *Status*

- Uklanja se *FlowEntry* (unos toka). *ForwardingRulesManager* će zatražiti SDN protokolni dodatak da ukloni tok (engl. *flow*) na mrežnom čvoru. U slučaju uspješne operacije, ažurirat će svoju bazu podataka i vratit će pripadajući statusni kod.

Import 2: ***import*** *org.opendaylight.controller.forwardingrulesmanager.FlowEntry*

- Predstavlja zahtjeve aplikacije toka na *ForwardingRulesManager* s ciljem postavljanja toka na mrežni čvor. *FlowEntry* sastoji se od toka, ciljanog mrežnog čvora, naziva toka i grupnog imena. Svi tokovi koji čine određenu politiku toka (puta) imaju isto grupno ime.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Metoda 1: ***Equals*** (*Object obj*)

Tip: *Boolean*

- Nadjačava *equals* u klasi *Object*.

Metoda 2: ***toString*** ()

Tip: *String*

- Nadjačava *toString* u klasi *Object*.

Metoda 3: ***getFlow*** ()

Tip: *Flow*

- Vraća *flow* sadržan u *FlowEntry-u*.

Metoda 4: ***getNode*** ()

Tip: *Node*

- Dohvaća mrežni čvor.

Paket *org.opendaylight.controller.hosttracker*

Import 1: ***import*** *org.opendaylight.controller.hosttracker.IfIptoHost*

- Ovo sučelje definira metode za dohvaćanje informacija o poznatim *Host*-ovima (krajnjim računalima). Također pruža metode za statičko postavljanje/uklanjanje *Host*-ova iz lokalne baze podataka.

Metoda 1: ***hostFind*** (*InetAddress addr*)

Tip: *HostNodeConnector*

- Aplikacije pozivaju metode ovog sučelja kako bi odredile povezanost IP adrese krajnjeg uređaja s MAC adresom te je li uređaj spojen s OpenFlow komutatorom. Te povezanosti se dodaju dinamički, ali mogu biti i dodane statički kroz *northbound* API. Ako je veza IP adrese i MAC adrese nepoznata, ARP zahtjev je odmah pokrenut kako bi se otkrio *Host*. Vraća se objekt razreda *HostNodeConnector* koji sadrži podatke o *Host-u*, tj. njegovu

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

MAC adresu, oznaku komutatora na koji je priključen, oznaku priključka, oznaku VLAN-a (engl. *Virtual Local Area Network*). Ako *Host* nije pronađen, vraća vrijednost *null*.

Paket *org.opendaylight.controller.hosttracker.hostAware*

Import 1: **import** *org.opendaylight.controller.hosttracker.hostAware.HostNodeConnector*

Metoda 1: ***getNodeConnector*** ()

Tip: *NodeConnector*

- Vraća *NodeConnector*.

Metoda 2: ***getNodeconnectorNode*** ()

Tip: *Node*

- Vraća *Node*.

Paket *org.opendaylight.controller.statisticsmanager*

Import 1: **import** *org.opendaylight.controller.statisticsmanager.IStatisticsManager*

- Sučelje koje definira dostupne metode za dohvaćanje statistika mrežnih čvorova.

Metoda 1: ***getFlowsNumber*** (*Node node*)

Tip: *Int*

- Vraća broj tokova postavljenih na komutatoru u kontekstu trenutne tablice tokova. Ako se kontekst odnosi na podrazumijevanu tablicu tokova, vraćena vrijednost je broj svih tokova postavljenih u komutatoru bez obzira na tablicu kojoj pripadaju. Vraća broj tokova na određenom čvoru ili (-1) ako čvor nije pronađen.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Metoda 2: ***getNodeConnectorStatistics*** (*NodeConnector nodeConnector*)

Tip: *NodeConnectorStatistics*

- Vraća statistike za navedeni *Node Connector* koji je preuzet od protokolnih dodataka komponente i spremljen od strane *Statistics Manager-a*. Vraća statistike *Node Connector-a* ili vrijednost *null* ako zatražene statistike nisu pronađene.

Paket *org.opendaylight.controller.switchmanager*

Import 1: ***import*** *org.opendaylight.controller.switchmanager.ISwitchManager*

- Glavna svrha ovog sučelja je pružiti metode za aplikaciju da može pristupiti različitim resursima sustava i popisu podataka koji uključuje čvorove, njihove priključke i njihova svojstva, konfiguraciju pravila prosljeđivanja mrežnog sloja, konfiguraciju čvora (engl. *Node*) te pozivom određenih metoda prikazuje mrežne uređaje (*Node*, *NodeConnector*, *NetworkDevices*) unutar aplikacije upravljačkog uređaja.

Metoda 1: ***getNetworkDevices*** ()

Tip: *List<Switch>*

- Vraća listu svih poznatih mrežnih uređaja u sustavu.

Metoda 2: ***getNodeConnector*** (*Node*, *String nodeConnectorName*)

Tip: *NodeConnector*

- Vraća *NodeConnector* preko njegovog imena.

Metoda 3: ***getNodeConnectorProp*** (*NodeConnector*, *String propName*)

Tip: *Property*

- Vraća specifično svojstvo *NodeConnector-a*.

Metoda 4: ***getNodeConnectors*** (*Node node*)

Tip: *Set<NodeConnector>*

- Vraća sve *NodeConnectore*. Status svakog *NodeConnectora* varira.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Metoda 5: ***getUpNodeConnectors*** (*Node node*)

Tip: *Set<NodeConnector>*

- Vraća sve *NodeConnectore* aktivne za dani čvor.

Import 2: ***import*** *org.opendaylight.controller.switchmanager.Switch*

- Razred opisuje informacije vezane uz komutator, koje uključuju MAC adresu, priključke i prikaz njega kao čvora.

Metoda 1: ***getNode*** ()

Tip: *Node*

- Vraća *Node*.

Metoda 2: ***getNodeConnectors*** ()

Tip: *Set<NodeConnector>*

- Vraća *NodeConnector-e*.

Paket *org.opendaylight.controller.topologymanager*

Import 1: ***import*** *org.opendaylight.controller.topologymanager.ITopologyManager*

- Sučelje koje pruža metode za interakciju s bazom podataka mrežne topologije.

Metoda 1: ***getNodeEdges*** ()

Tip: *Map<Node,Set<Edge>>*

- Vraća nepromijenjenu mapu indeksiranu čvorom (engl. *Node*) i javlja sve poveznice koji ulaze u čvor ili izlaze iz čvora.

Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

3.3 Upute za instalaciju i konfiguraciju postavki upravljačkog uređaja OpenDaylight

Unutar ovog poglavlja opisan je postupak instalacije OpenDaylight upravljačkog uređaja verzije Lithium [20] iz izvornog koda na operacijskom sustavu Linux Mint (17.2). Instalacija se odvija u 3 glavna koraka:

1. Instalacija ovisnosti potrebnih za sljedeće korake
2. Instalacija Maven alata
3. Instalacija upravljačkog uređaja

3.3.1 Instalacija ovisnosti

Budući da je Linux Mint distribucija OS-a Linux zasnovana na Debian sustavu, instalacija ovisnosti *pkg-config*, *gcc*, *make*, *ant*, *g++*, *git*, *libboost-dev*, *libcurl4-openssl-dev*, *libjson0-dev*, *libssl-dev*, *openjdk-7-jdk*, *unixodbc-dev*, *xmlstarlet*, jednostavna je uz pomoć APT alata za dohvaćanje i instalaciju potrebnih paketa. Tako će se ovisnosti instalirati sljedećom naredbom u ljesci:

```
$ apt-get install pkg-config gcc make ant g++ git libboost-dev libcurl4-openssl-dev libjson0-dev libssl-dev openjdk-7-jdk unixodbc-dev xmlstarlet
```

3.3.2 Instalacija Maven alata

OpenDaylight zahtjeva Maven verziju 3.3.1 ili noviju koja nije dostupna u službenim Linux Mint repozitorijima, pa je potrebno dohvatiti najnoviju verziju s Apache stranice:

<https://maven.apache.org/>

Dovoljno je dohvatiti binarne datoteke zapakirane u *.tar.gz* formatu, koje se raspakiraju sljedećom naredbom u ljesci pozicioniranoj u direktoriju u kojem je dohvaćena arhiva pohranjena (*x.y.z* je verzija alata koja je dohvaćena):

```
$ tar zxvf apache-maven.x.y.z.tar.gz /usr/local/
```

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Nakon toga je samo potrebno dodati izvršne datoteke u PATH varijablu okruženja:

```
$ export PATH=/usr/local/apache-maven.x.y.z/bin:PATH
```

Budući da OpenDaylight održava vlastite repozitorije (ne nalaze se u Maven Central repozitoriju), Maven nema informacija o potrebnim artefaktima za izgradnju. Također, kako je OpenDaylight organiziran kao skup međuovisnih projekata, izgradnja jednog od njih obično zahtjeva dohvat artefakata iz ostatka skupa. Kako bi se ovo omogućilo, Maven instalacija mora znati i „naučiti“ lokaciju svih OpenDaylight repozitorija. Taj problem se najlakše rješava sljedećim naredbama:

```
$ cp -n ~/.m2/settings.xml{,.orig}
$ wget -q -O - https://raw.githubusercontent.com/opendaylight/odlparent/master/settings.xml >
~/.m2/settings.xml
```

3.3.3 Instalacija upravljačkog uređaja

Budući da se od Helium verzije nadalje OpenDaylight upravljački uređaj izvodi unutar Karaf okruženja[21], potrebno je dohvatiti tu distribuciju upravljačkog uređaja iz integracijskog projekta OpenDaylight sljedećom naredbom:

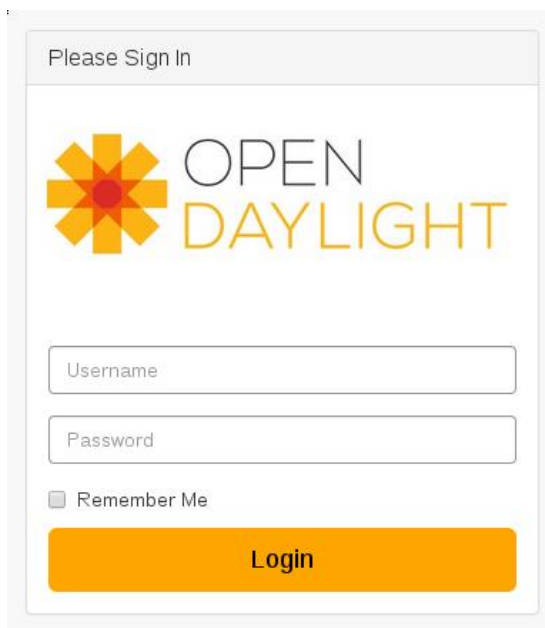
```
$ git clone https://git.opendaylight.org/gerrit/p/integration/distribution.git
```

Pozicionirajmo se unutar dohvaćenog direktorija te izgradimo upravljački uređaj:


```
$ cd distribution
$ mvn clean install -D skipTests
```

Za kraj ostaje jedino pokrenuti upravljački uređaj unutar Karaf okruženja:

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Please Sign In

 OPEN DAYLIGHT

Username

Password

☐ Remember Me

Login

Slika 3-5 Obrazac za prijavu na grafičko sučelje platforme OpenDaylight, verzija Lithium

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

4. Integracija upravljačkog uređaja OpenDaylight u laboratorijsku mrežu

Kao algoritam usmjeravanja korišten je algoritam optimizacije kolonijom mrava koji se svrstava u algoritme zasnovane na inteligenciji roja, jer se temelji na razmjeni informacija među jedinkama. Programsko ostvarenje algoritma izvedeno je kao proširenje upravljačkog uređaja *OpenDaylight* [1].

Osnovni koncept algoritma za usmjeravanje u mrežama zasnovanog na kvaliteti usluge je usmjeravanje paketa najboljim mogućim putem kroz određenu mrežu s obzirom na vrstu višemedijske usluge koja se želi pružiti i karakteristike mreže koja se koristi. Kvaliteta usluge (engl. *Quality of service*, QoS) predstavlja ukupnu učinkovitost telefonske ili računalne mreže koja je vidljiva od strane korisnika istih. Za kvalitetno mjerenje kvalitete usluge potrebno je uzeti u obzir nekoliko standardnih aspekata mreže, kao što su vjerojatnost pogrešaka, brzina prijenosa, propusnost poveznice, kašnjenje u prijenosu, dostupnosti i drugi. Za određene vrste višemedijskih usluga gledaju se određene karakteristike mreže, pa se algoritmom žele pronaći putevi koji pružaju najbolje karakteristike za svaku vrstu usluga. Te karakteristike u SDN mreži upravljački uređaj prikuplja periodički od svih aktivnih mrežnih uređaja u mreži. Također, on vodi računa o topologiji mreže tako što prima poruke od svih mrežnih uređaja koji javljaju svoju prisutnost i podatke o svojim susjedima.

Zahtjev za prijenosom paketa mrežom zaprima upravljački uređaj i na temelju odabrane vrste višemedijske usluge pronalazi najoptimalniji put za tu vrstu usluge. Usluga, odnosno njezini medijski tokovi mogu biti vrste video, audio ili općeniti podatak.

4.1 Programsko proširenje izvornog algoritma

U sklopu programskog rješenja algoritma opisanog u prethodnom poglavlju ostvareno je proširenje algoritma pomoću kojeg je moguće uračunati gubitak paketa na poveznicama između mrežnih uređaja. Osim toga, kao dodatak izvornom kodu implementirana je metoda *installPredefinedFlows()* koja za sve mrežne uređaje ugrađuje unaprijed definirane tokove, odnosno unosi zapis u tablicu koja omogućuje obradu DHCP zahtjeva i odgovora. Programski

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

kod navedene metode nalazi se u dodatku B. Također, proširena je funkcionalnost metode *recieveDataPacket* koja kao ulazni parametar prima paket u *RawPacket* formatu. Proširena funkcionalnost odnosi se na klasifikaciju paketa prema njihovoj vrsti (*Ethernet*, *IEEE8021Q*, *ARP*, *IPv4*, *ICMP*) koja se vrši dekapsuliranjem paketa s VLAN zaglavljem te dodavanjem novog krajnjeg uređaja (engl. host) unutar baze podataka *OpenDaylight-a* ukoliko je primljeni paket tipa *ARP*.

4.1.1 Funkcijski zahtjevi

Izračun gubitka paketa na poveznicama potrebna je zbog točnosti ukupnog gubitka paketa između dva čvora u mreži, jer se paketi mogu gubiti i na samoj poveznici, a ne samo u mrežnim uređajima. Taj gubitak može se dogoditi zbog, primjerice, smetnji u poveznici. Osim toga, potrebno je implementirati dinamičko dodavanje novih krajnjih uređaja primitkom *ARP* zahtjeva te točan prikaz topologije mreže kao i implementirati funkcionalnost koja će omogućiti *ping* (međusobnu provjeru dostupnosti) između dva računala unutar laboratorijske mreže.

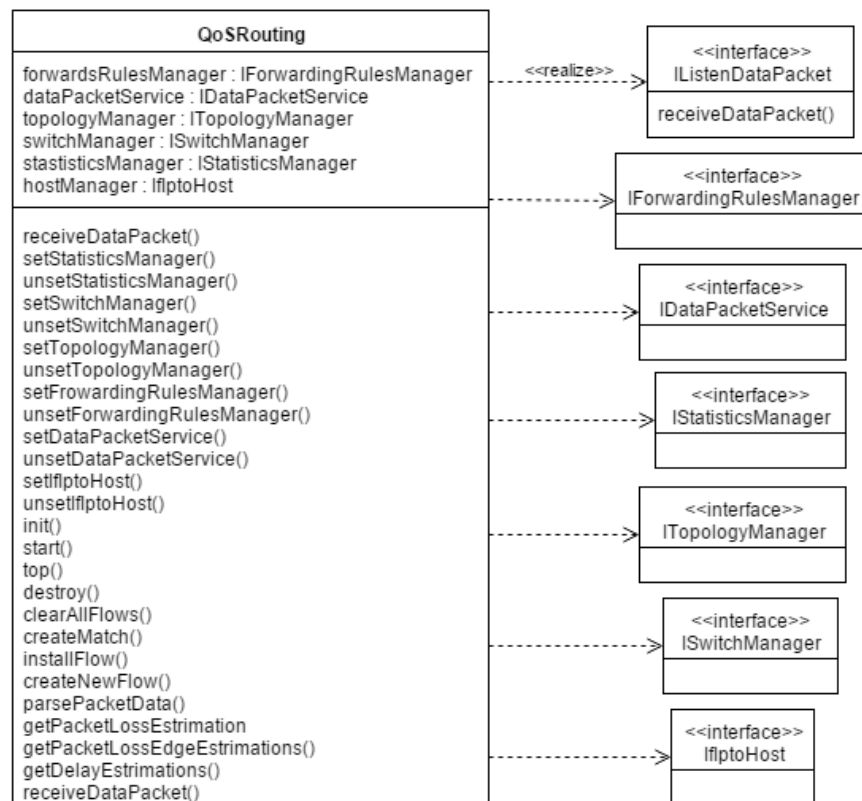
4.1.2 Dijagrami razreda

Implementiranjem opisanog proširenja izmijenjene su klase *QoSRouting* i *AntColony*, na *Slika 4-1* i *Slika 4-2* prikazani su dijagrami razreda te dvije klase.

U *QoSRouting* klasi je izvedena nova metoda nazvana *getPacketLossEdgeEstimations* koja obavlja proračun gubitka paketa na poveznicama između čvorova i vraća ga pomoću mape u kojoj se nalaze sve poveznice i procjena gubitka paketa na njima.

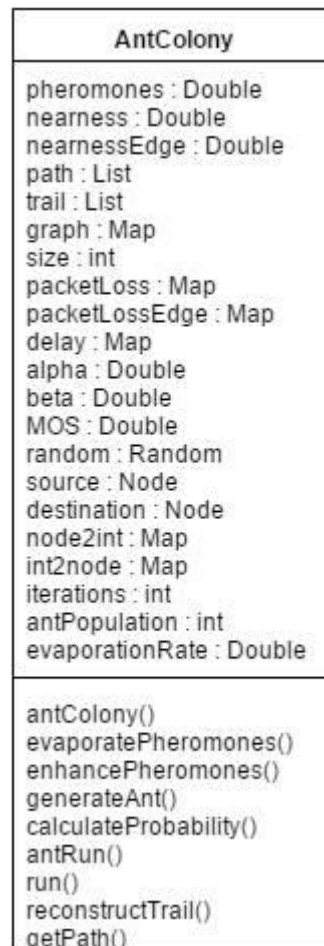
Ta procjena se šalje u *AntColony* u kojoj su dodani atributi *packetLossEdge* i *nearnessEdge* te petlja koja iz *packetLossEdge* puni *nearnessEdge* i pribraja taj gubitak paketa na poveznicama kašnjenju na samim čvorovima.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Slika 4-1 Dijagram razreda QoSRouting s metodom za izračun gubitaka na poveznici

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Slika 4-2 Dijagram razreda AntColony s mapom za pohranu gubitaka paketa na poveznicama

4.1.3 Programsko proširenje

To proširenje ostvareno je pomoću dodatne metode unutar razreda *QoSRouting* nazvane *getPacketLossEdgeEstimations*, koja iz upravljačkog uređaja dohvaća sve poveznice u mreži i za svaku poveznicu izračuna taj gubitak paketa kao postotak paketa koji su primljeni u mrežnom uređaju nakon poveznice od ukupnih paketa koji su poslani iz mrežnog uređaja kroz tu poveznicu. Na *Slika 4-3* prikazan je isječak programskog koda implementiranog proširenja za izračun gubitka na poveznicama.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

public <Edge, Double> getPacketLossEdgeEstimations() {
    Map<Edge, Double> ret = new HashMap<Edge, Double>();
    Map<Node, Set<Edge>> edges = this.topologyManager.getNodeEdges();
    for (edge e: edges.keySet()) {
        NodeConnector i = e.getTailNodeConnector();
        NodeConnector j = e.getHeadNodeConnector();
        NodeConnectorStatistics stati = this.statisticsManager.getNodeConnectorStatistics(i);
        NodeConnectorStatistics statj = this.statisticsManager.getNodeConnectorStatistics(j);

        loss *= (stati.getReceivePacketCount() / statj.getTransmitPacketCount());
        ret.put(e, 1 - loss);
    }
    return ret;
}

```

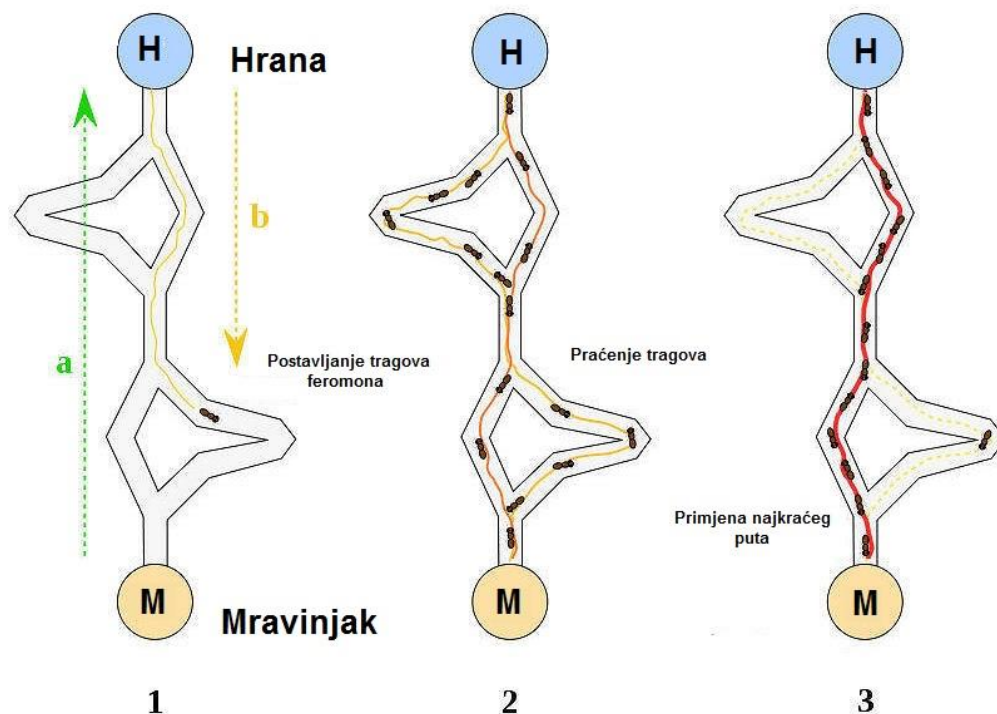
Slika 4-3 Isječak programskog koda za izračun gubitka na poveznicama

Statistika se sprema u mapu i šalje kao argument klasi *AntColony* u kojoj se gubitak paketa na poveznicama pribraja gubitku paketa na samim čvorovima, tako da se sada kao gubitak paketa između dva mrežna uređaja dobiva produkt gubitaka paketa na početnom čvoru poveznice, poveznici i krajnjem čvoru poveznice.

4.2 Algoritam usmjeravanja zasnovan na optimizaciji kolonijom mrava

Algoritam optimizacije kolonijom mrava (engl. Ant Colony Optimization, ACO) prvi je spomenuo Marco Dorigo 1991. u tezi o optimizaciji[19], učenju i prirodnim algoritmima. Temelji se na oponašanju mrava u prirodi koji u potrazi za hranom za sobom ostavljaju tragove feromona, a ostali mravi ih mogu osjetiti i tako naći put do hrane. Ako negdje osjete jači trag feromona, veća je vjerojatnost da će krenuti tim putem. Razmjena informacija među mravima ne događa se direktno, već indirektno, modifikacijom okoline feromonima. Snaga feromonskog traga ovisi o brzini isparavanja feromona i o kvaliteti izvora hrane. Osnovni koncept algoritma usmjeravanja zasnovanog na optimizaciji kolonijom mrava prikazan je na *Slika 4-4*.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Slika 4-4 Kretanje mrava u potrazi za hranom

4.2.1 Osnovni koncept algoritma

ACO algoritam je kombinacija *a priori* i *a posteriori* razmišljanja virtualnih mrava. *A priori* dio mrava bira put ne znajući ništa o već prikupljenim informacijama o putevima, tj. zanemaruje feromone, dok *a posteriori* dio mrava ovisi o feromonima, tj. informacijama skupljenima prije njega. To je u algoritmu ostvareno pomoću parametara α i β . Parametar α nazivamo diverzifikacija te on pojačava *a priori* razmišljanje mrava i time pomaže da se istraži što više različitih puteva. Parametar β nazivamo intenzifikacija i on pojačava *a posteriori* razmišljanje mrava, pa se detaljnije pretražuju rješenja koja će vjerojatnije imati bolje rezultate.

Algoritam prvo učitava parametre, među kojima su i kašnjenje i vjerojatnost gubitka paketa dobiveni iz mreže, i inicijalizira strukture podataka, zatim šalje populaciju virtualnih mrava kroz simuliranu mrežu u onoliko iteracija koliko je unaprijed definirano. Tokom prolaska

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

kroz mrežu, mrav čuva listu čvorova koje je već posjetio, a sljedeći čvor u koji mrav ide određuje se pomoću vjerojatnosti koje se izračunavaju za sve susjedne čvorove na temelju parametara α i β . Nakon što je cijela populacija mrava prošla kroz mrežu, osvježavaju se vrijednosti feromona, koje se prvo ishlapljaju stopom koja je definirana u parametrima algoritma, a zatim mravi koji su prošli graf pojačavaju tragove feromona ovisno o kvaliteti usluge pojedinih pronađenih puteva. Nakon onoliko iteracija koliko je definirano, bira se mrav s najvećom kvalitetom usluge za traženu vrstu usluge i vraća se kao konačno rješenje.

Kašnjenje i gubitak paketa mora biti skalarno izraženo za uporabu u algoritmu. Kašnjenje je aditivna mjera pa se kašnjenje od kraja do kraja mreže računa kao suma kašnjenja na svim čvorovima na putu. Dok je gubitak paketa multiplikativna mjera, gubitak paketa s kraja na kraj mreže računa se kao produkt gubitka paketa na svakom čvoru.

4.2.2 Integracija algoritma u upravljački uređaj OpenDaylight

Algoritam je napisan programskim jezikom *Java* kao i upravljački uređaj *OpenDaylight*, a za izradu proširenja korištena je *Hydrogen* verzija upravljačkog uređaja *OpenDaylight*. Opis inicijalne izvedbe algoritma usmjeravanja dan je u [1].

Algoritam je ostvaren tako da se prilikom dolaska paketa do upravljačkog uređaja, tj. do proširenja *QoSRouting* koje implementira opisani algoritam, stvara novo pravilo usmjeravanja na mrežnim uređajima koji se nalaze na najkraćem putu od klijenta do poslužitelja. Proširenje *QoSRouting* informacije dobiva od sučelja implementiranih u upravljačkom uređaju *OpenDaylight*. Neka od bitnih sučelja koja programsko rješenje koristi su:

- *ISwitchManager*, koje služi za dohvrat mrežnih uređaja i vodi računa oko ispada i dodavanja mrežnih uređaja;
- *ITopologyManager*, koje služi za dohvrat trenutne topologije mreže;
- *IStatisticsManager*, koje služi za dohvrat potrebnih statističkih podataka s mrežnih uređaja;
- *IForwardingRulesManager*, koje služi za upravljanje nad tablicama prosljeđivanja (tokova) mrežnih uređaja;
- *IDataPacketService*, koje služi za enkapsulaciju ili dekapulaciju paketa; te
- *IIptToHost*, koje služi za dohvrat IP adresa krajnjih računala spojenih na mrežne uređaje.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Inicijalizacija svih sučelja te njihove *get* i *set* metode ostvarene su u klasi *Activator*. Kako bi ovo proširenje moglo dohvatiti pakete koji su poslani upravljačkom uređaju, potrebno je implementirati sučelje *IListenDataPacket* i nadjačati metodu *receiveDataPacket*.

Procjena kašnjenja i vjerojatnost gubitka paketa na mrežnim uređajima izračunava se pomoću sučelja *IStatisticsManager*, koje periodički prikuplja statističke podatke i vraća ih pozivom metode *getNodeConnectorStatistics*, ali nema podatke o kašnjenju. Kašnjenje je procijenjeno produktom broja zapisa u tablici tokova mrežnog uređaja i vremena potrebnog za prijenos jedne veličine IP datagrama. Pri izračunu vremena koristi se brzina poveznice mrežnog uređaja koja se dobije od upravljačkog uređaja. vjerojatnost gubitka paketa računa se direktno iz statističkih podataka dobivenih iz sučelja *IStatisticsManager* kao suma paketa poslanih s određenog sučelja mrežnog uređaja podijeljena sa sumom paketa primljenih na tom istom sučelju.

U razredu *AntColony* ostvaren je cijeli algoritam optimizacije kolonijom mrava, a za sve tri vrste medijskih tokova ostvaren je poseban razred mrava koji implementiraju bazni razred *Ant* te svaka vrsta mrava nadjačava metodu za izračun kvalitete usluge, ovisno za koju se vrstu medijskog toka pojedina vrsta mrava koristi.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

5. HTTP aplikacijsko sučelje za upravljanje kvalitetom višemedijskih usluga

Aplikacijsko programsko sučelje (engl. *Application program interface*, API) čini skup pravila, protokola i alata za izradu programskih aplikacija. Općenito, API predstavlja programsku komponentu u smislu njezinih operacija, ulaza, izlaza, osnovnih tipova, definirajući tako funkcionalnosti koje su nezavisne o njihovom načinu implementacije [2]. Osim toga, API definira način komunikacije između programskih komponenti. Iz tog razloga, postojanje dobro oblikovanog API-ja razvojnim inženjerima aplikacija olakšava posao pružajući sve „građevne blokove“.

U sklopu ovoga projekta cilj je bio izraditi API koji će omogućiti povezivanje aplikacijskog i upravljačkog sloja SDN mreže za potrebe upravljanje kvalitetom višemedijskih usluga. Za izradu istoga potrebno je zadovoljiti određene funkcijske zahtjeve, ali i zahtjeve same domene primjene. Definicije zahtjeva nastale su analizom sličnih API-ja te potrebama projektnog zadatka. Neki od zahtjeva su: mogućnost prijenosa parametara višemedijskih usluga u unaprijed definiranom formatu, funkcijska podjela API-ja na klijentsku i poslužiteljsku stranu, izvedba API-ja pomoću REST arhitekturnih obrazaca ispunjavanje načela arhitekture i komunikacije postavljenih od strane HTTP protokola.

5.1 Arhitektura klijent-poslužitelj

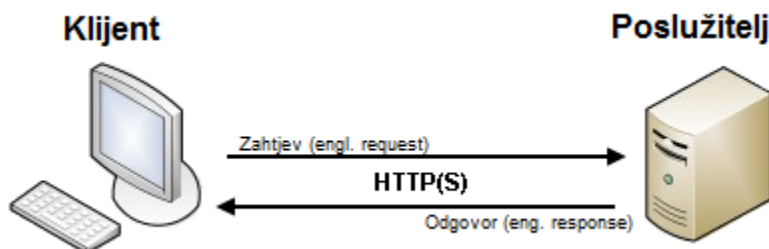
Arhitektura klijent-poslužitelj predstavlja arhitekturi obrazac koji se temelji na raspodjeli posla između pružatelja određene usluge ili resursa, poslužitelja, i zahtjevatelja usluge, klijenta. Ona je jedan od primjera raspodijeljenih sustava čije su značajke:

- obradu podataka i izračunavanje obavljaju odvojeni programi;
- uobičajeno su ti odvojeni programi na odvojenom sklopovlju (računalima, čvorovima, stanicama); te
- odvojeni programi međusobno komuniciraju računalnom mrežom.

Osim toga, važno je primijetiti da, kako bi komunikacija uopće bila moguća, klijent i poslužitelj moraju „govoriti istim jezikom“, tj. koristiti unaprijed poznati format podataka i komunikacijski protokol, neovisno o njihovim (najčešće različitim) sklopovskim i programskim izvedbama.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Na *Slika 5-1* prikazan je osnovni koncept komunikacije između klijenta i poslužitelja putem HTTP(s) protokola.



Slika 5-1 Model klijent-poslužitelj

Detaljan prikaz ostvarenja veza između klijenta i poslužitelja te daljnje komunikacije prikazan je sljedećom sekvencom aktivnosti:

Sekvenca aktivnosti rada arhitekture klijent-poslužitelj:

1. *Poslužitelj započinje s radom*
2. *Poslužitelj čeka na dolazak klijentskog zahtjeva (poslužitelj sluša)*
3. *Klijenti započinju s radom*
4. *Klijent pokušava spajanje na poslužitelja*
5. *Poslužitelj omogućava spajanje klijenta (ako su ispunjeni uvjeti/želi)*
6. *Klijent šalje poruke/zahtjeve prema poslužitelju*
7. *Poslužitelj poduzima potrebne akcije za pristigle poruke od strane klijenta*
8. *Poslužitelj šalje povratnu poruku klijentu nakon obrade zahtjeva*
9. *Klijent i poslužitelj nastavljaju s radom (do prekida ili prestanka rada)*

Iako ovakav pristup arhitekturi ima mnoge prednosti kao što su podjela posla, udaljeno pristupanje funkcionalnostima poslužitelja, centralizirana pohrana podataka, jednostavnost entiteta, odvojeno oblikovanje itd., on također ima i svoje nedostatke. Najizraženiji problemi koji su i danas jedna od tema rasprave su skalabilnost, sigurnost i održavanje. Sigurnost predstavlja veliki problem upravo iz razloga što je teško razviti savršenu enkripciju i vatrozidove (engl. *firewall*) koji će štiti podatke od ilegalnih pristupa. Održavanje predstavlja problem upravo zbog načina odvojenog oblikovanja klijenta i poslužitelja. Iz tog razloga sva programska potpora mora

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

biti kompatibilna prema unatrag (engl. *backwards compatibility*) te kompatibilna s drugim verzijama klijenata i poslužitelja.

Neki od primjera arhitekture klijent-poslužitelj su usluge WWW (engl. *World Wide Web*), elektronička pošta (engl. *e-mail*) i mnoge druge.

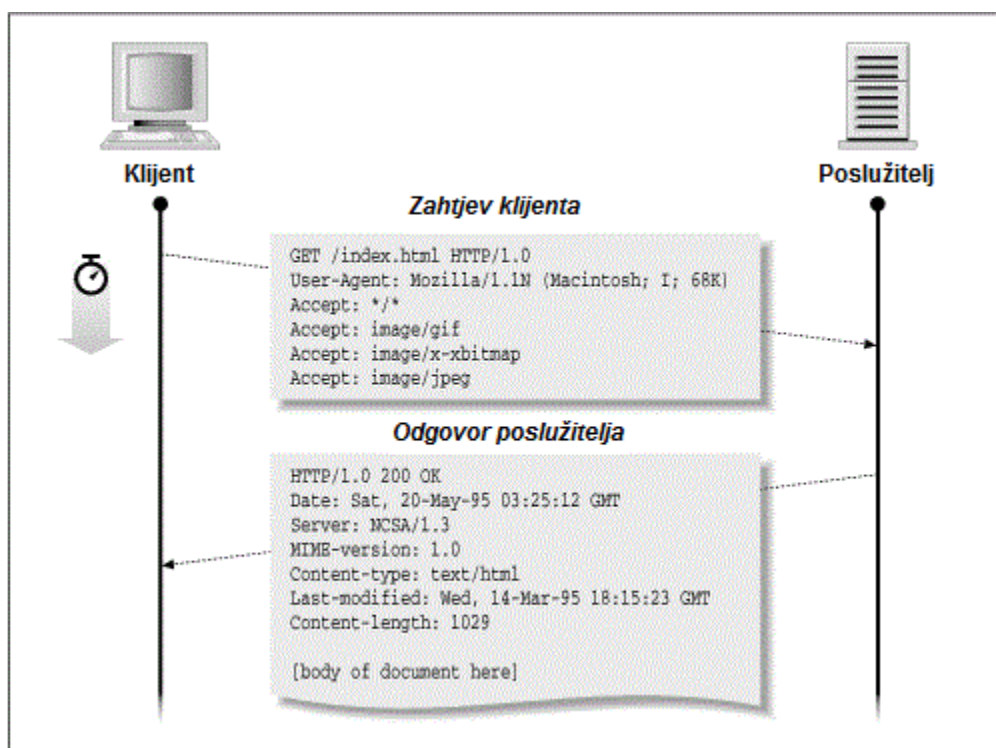
5.2 Osnovni koncept protokola HTTP

HTTP (engl. *Hypertext Transfer Protocol*) predstavlja internetski protokol aplikacijskog sloja koji definira format i način razmjene poruka između klijenta i poslužitelja. HTTP protokol se može izvesti u bilo kojoj mreži koja se temelji na internetskim protokolima, a može prenositi različite vrste podataka. Prva verzija HTTP protokola pojavila se 1990. godine u obliku 0.9 sa jednostavnom funkcionalnošću prijenosa samo dokumenata temeljenih na hipertekstu. Tijekom sljedećih godina razvile su se verzije 1.0, odnosno danas važeća 1.1, koja je krajem 2010. godine specificirana u RFC-u 2616.

HTTP definira dvije vrste poruke, a to su zahtjev (engl. *request*) i odgovor (engl. *response*). Zahtjev definira operacije, resurse i verziju protokola HTTP, dok odgovor sadrži ishod zahtjeva koji je opisan statusnim kodom, i ovisno o vrsti zahtjeva i ishodu, sadržaj traženog resursa.

U okviru ovog projekta komunikacija između klijenta i poslužitelja temelji se na razmjeni HTTP GET i POST zahtjeva te odgovora 200 OK. GET metoda služi za dohvaćanje resursa te se aktivira upisivanjem adrese u web preglednik ili klikom na određeni link. Ako poslužitelj sadrži resurs, zatražen od strane klijenta, vraća ga u tijelu odgovora, inače vraća pogrešku. Druga metoda, POST, služi za „aktiviranje“ resursa. Drugim riječima, podaci sadržani u tijelu HTTP POST metode šalju se poslužitelju na obradu. Na ovaj način u okviru ovoga projekta implementacija klijenta, *SimpleHttpClient*, unutar tijela POST poruke poslužitelju *SimpleHttpServer* šalje parametre višemedijske sjednice na daljnju obradu. Na *Slika 5-2* prikazan je format zahtjeva i odgovora GET metode u komunikaciji između klijenta i poslužitelja.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Slika 5-2 Primjer razmjene HTTP poruka

5.3 Model HTTP API-ja za upravljanje kvalitetom višemedijskih usluga

Kao što je navedeno, implementirani model HTTP API-ja ima namjenu povezivanja aplikacijskog i upravljačkog sloja SDN mreže. Točnije, ima osnovnu zadaću prijenosa parametara višemedijskih usluga od SDN aplikacija prema upravljačkom uređaju. Model se sastoji od dva osnovna funkcionalna dijela, a to su klijentski i poslužiteljski dio. Ovakvom podjelom funkcionalnosti zadovoljen je princip REST arhitekture.

Izvedeni klijentski dio API-ja sadrži funkciju čitanja tekstualne datoteke unutar koje su sadržani parametri sjednice višemedijske usluge te prosljeđivanja istih unutar tijela POST zahtjeva prema poslužiteljskom dijelu API-ja. Na poslužiteljskoj strani implementirane su funkcije pokretanja poslužitelja na unaprijed definiranoj IP adresi i transportnim vratima, kao i onima ručno unesenim. Također, poslužiteljski dio API-ja sadrži mogućnosti obrade primljenih HTTP GET i POST zahtjeva, pri čemu se pretpostavlja da GET zahtjevi sadrže prazno tijelo poruke. Prilikom primljenog POST zahtjeva poslužiteljski dio pokreće određene funkcije za

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

obradu tijela poruke, nakon čega klijentu vraća odgovor ovisno o ishodu. Za potrebe programske izvedbe modela aplikacijskog programskog sučelja u ovome projektu, opisi sjednica višemedijskih usluga zapisani su i strukturirani u obliku običnog teksta (engl. *plain text*).

5.4 Programska izvedba HTTP API-ja

Programska izvedba modeliranog aplikacijskog programskog sučelja ostvarena je pomoću programskog jezika Java. Razlog uporabe upravo ovog programskog jezika jest njegova popularnost unutar krugova razvojnih inženjera mrežnih aplikacija, pa tako i SDN-a. Također, zbog izvedbe upravljačkog uređaja OpenDaylight u programskom jeziku Java, logičan korak bio je izvesti samo sučelje unutar istoga okruženja kako bi se olakšala integracija. S obzirom da postoji više načina za izradu HTTP API-ja, tj. jednostavnog web poslužitelja, a OpenDaylight zajednica nije pružila detaljnije informacije o mogućnosti uvoza stranih knjižnica unutar strukture upravljačkog uređaja, odlučeno je da se izrade dva web poslužitelja koristeći standardne i nestandardne Java biblioteke kako bi testirali kompatibilnost istih sa samim OpenDaylight upravljačkim uređajem.

5.4.1 Funkcijski zahtjevi

Prije same programske izvedbe HTTP API-ja nameću se određeni funkcijski zahtjevi ključni za ispravnu funkcionalnost sustava. Važno je naglasiti kako je većina funkcijskih zahtjeva unaprijed definirana samom domenom primjene. Neki od zahtjeva su:

- sustav mora biti temeljen na REST arhitekturi;
- mogućnost ručne konfiguracije poslužitelja (određivanje načina njegovog pokretanja)
 - a) poslužitelj se nalazi (sluša) na unaprijed definiranim vratima i *localhost* adresi računala
 - b) poslužitelj se nalazi na ručno definiranim vratima i *localhost* adresi računala
 - c) poslužitelj se nalazi na ručno definiranim vratim i IP adresi, *npr. 192.168.1.70:8080*;
- parametri sjednice višemedijskih usluga nalaze se u točno definiranom formatu i u obliku tekstualne datoteke;

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

- mogućnost slanja GET i POST zahtjeva prema poslužitelju;
- klijentski dio API-ja mora imati mogućnost čitanja parametara sjednice iz datoteke te ugnježđivanje istih unutar HTTP poruke POST zahtjeva;
- obrada zahtjeva u skladu s vrstom poruke (GET, POST);
- mogućnost obrade i spremanje bilo kojeg broja sjednica unutar kojih se nalazi proizvoljan broj medijskih tokova tipa video, audio i ostalo (podaci);
- poslužitelj mora vratiti ishod obrade zahtjeva klijentu; te
- otpornost sustava na pogreške, uz obavijest korisnika o vrsti pogreške nastale pri komunikaciji.

Na *Slika 5-3* prikazan je format opisa sjednice višemedijskih usluga korišten unutar implementacije API-ja.

sessionId=17565501092016	sessionId = jedinstveni identifikator sjednice
m=video	m = vrsta medija
ip_src=192.168.31.1	ip_src = izvorišna adresa
ip_dst=192.168.1.1	ip_dst = odredišna adresa
proto=udp	proto = korišteni transportni protokol
port_src=22222	port_src = izvorišna vrata
port_dst=33333	port_dst = odredišna vrata
codec=mp4	codec = tip kodeka medija
bitrate=5600	bitrate = bitrate medija
m=audio	
ip_src=192.168.1.1	
ip_dst=192.168.31.1	
proto=udp	
port_src=44444	
port_dst=55555	
codec=mp3	
bitrate=128	
m=data	
ip_src=192.168.1.1	
ip_dst=192.168.31.1	
proto=tcp	
port_src=11111	
port_dst=22222	
bitrate=10000	

Slika 5-3 Format opisa sjednice višemedijske usluge

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

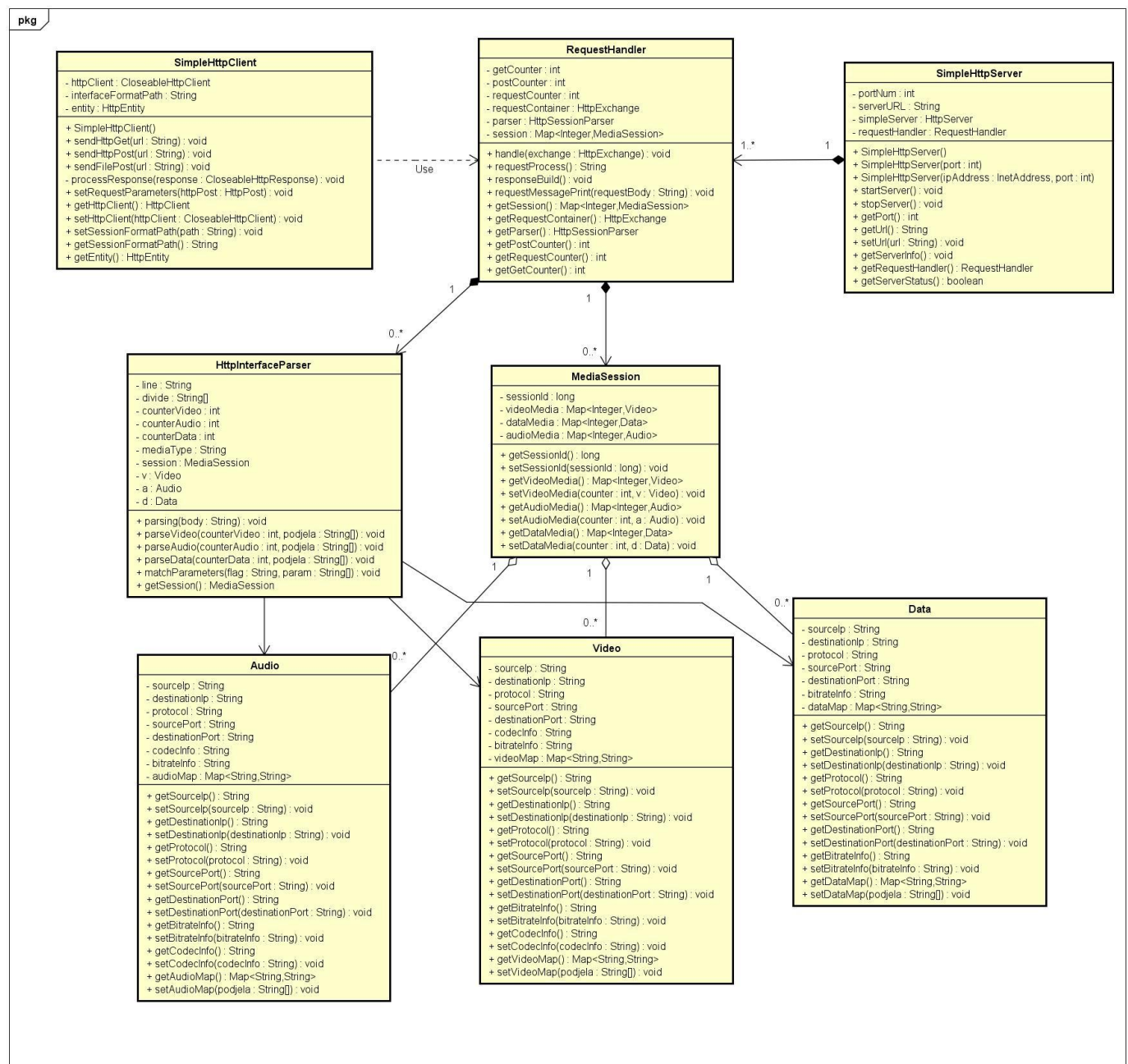
5.4.2 Struktura programske izvedbe HTTP API-ja izrađenog korištenjem nestandardnih Java biblioteka

HTTP API implementiran je koristeći nestandardne Java biblioteke (engl. *library*). Navedene biblioteke nalaze se unutar *com.sun.net.httpserver* paketa. Struktura programske izvedbe API-ja sastoji se od 2 temeljna dijela, *HttpClient* i *HttpServer*. Funkcionalnost klijenta ostvarena je pomoću razreda *SimpleHttpClient*, dok je funkcionalnost poslužitelja ostvarena pomoću razreda *SimpleHttpServer*, *RequestHandler*, *HttpInterfaceParser* te razreda koji služe kao spremnici (engl. *placeholder*) parametara pojedine sjednice, *MediaSession*, *Video*, *Audio*, *Data*. Detaljnije funkcionalnosti i sadržaj pojedinih razreda API-ja prikazane su kroz dijagram razreda, objekata te sekvencijske dijagrame, u sljedećim poglavljima, napravljenim u jeziku UML (engl. *Unified Modeling Language*).

5.4.3 Dijagram razreda s opisom

Na *Slika 5-4* prikazan je dijagram razreda API-ja modeliranog koristeći nestandardne Java biblioteke unutar *com.sun.net.httpserver* paketa.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Slika 5-4 Dijagram razreda modeliranog API-ja

Razred *SimpleHttpClient* sadrži tri atributa koji služe za instanciranje objekta tipa *CloseableHttpClient*, koji izvodi sam rad HTTP klijenta, put to datoteke sa opisom parametara sjeđnice višemedijskih usluga, te entitet koji služi kao spremnik za zahtjev koji se šalje poslužitelju. U okviru projekta napravljen je i poseban statičan razred *Launcher* kao testna

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

komponenta sa zadaćom pokretanja klijentskog i poslužiteljskog dijela. Kao ulazni argument pri pokretanju, razred *Launcher* prima put to datoteke s opisom sjednice. Osim toga, razred *SimpleHttpClient* sadrži konstruktor *SimpleHttpClient()* koji služi za instanciranje objekta tipa *CloseableHttpClient* pomoću metode *setHttpClient(CloseableHttpClient httpClient)*. Dodatno, razred sadrži i metode *sendHttpGet(String url)*, tj. *sendHttpPost(String url)*, koje služe za slanje HTTP GET, odnosno POST zahtjeva prema poslužitelju. Također, razred sadrži i metode *processResponse(CloseableHttpResponse response)* i *setRequestParameters(HttpPost httpPost)* koje služe za obradu odgovora od strane poslužitelja te postavljanje tijela POST metode sadržajem datoteke s opisom sjednice. Dodatno, razred sadrži određene *get* i *set* metode koje su nužne sa ispravno funkcioniranje razreda *SimpleHttpClient*.

Razred *SimpleHttpServer* zadužen je za stvaranje, pokretanje i zaustavljanje HTTP poslužitelja. Sadrži četiri razreda. Prvi, *simpleServer*, služi za referenciranje objekta razreda *HttpServer*, koji predstavlja poslužitelja, dok atributi *portNum* te *serverUrl* služe za spremanje vrata i mrežne adrese na kojima se poslužitelj pokreće te atribut koji predstavlja instancu razreda namjenjenu za prihvata zahtjeva od klijenta koju mu poslužitelj prosljeđuje. Također, razred *SimpleHttpServer* sadrži tri konstruktora izgrađenih na principu polimorfizma, kako bi se omogućilo instanciranje, odnosno pokretanje poslužitelja prema navedenim funkcijskim zahtjevima. Unutar navedenih konstruktora implementira se sučelje tipa *RequestHandler*, koje služi za primanje zahtjeva od strane klijenta, kao i *URI* (npr. *192.168.1.70:8080/get*) na kojem se ono nalazi. Na *Slika 5-5* prikazan je programski isječak koji implementira navedenu funkciju. Osim toga, razred *SimpleHttpServer* sadrži metode *startServer()* i *stopServer()* koje služe za pokretanje, odnosno zaustavljanje rada instanciranog poslužitelja. Dodatno, razred sadrži *get* i *set* metode koje služe za spremanje i dohvaćanje navedenih atributa.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

60
61 // Create an instance of HTTP server object on custom address and port
62 // (entered by user)
63 public SimpleHttpServer(InetAddress ipAddress, int port) throws IOException, BindException, NullPointerException {
64     try {
65         simpleServer = HttpServer.create(new InetSocketAddress(ipAddress, port), 0);
66         System.out.println("Server has started on: " + ipAddress.toString() + ":" + port);
67         simpleServer.createContext("/get", new RequestHandler());
68         simpleServer.setExecutor(null);
69         this.portNum = port;
70         this.serverURL = ipAddress.toString();
71         startServer();
72     } catch (BindException e) {
73         System.out.println("Server cannot bind to selected port and IP address!");
74         printCreateError();
75     } catch (NullPointerException e) {
76         System.out.println("IP address value: null !");
77         printCreateError();
78     } catch (IOException e) {
79         printCreateError();
80     }
81 }
82

```

Slika 5-5 Programski isječak instanciranja HTTP poslužitelja i funkcije za prijem zahtjeva (RequestHandler)

Nakon što HTTP poslužitelj primi određeni zahtjev (u našem slučaju GET ili POST), on se šalje na daljnju obradu. Tu zadaću ostvaruju atributi i metode unutar razreda *RequestHandler*. Razred *RequestHandler* sadrži metodu *handle(HttpExchange exchange)* koja obavlja funkciju prijema i obrade zahtjeva te generiranja povratne poruke. Osim toga, razred *RequestHandler* sadrži određene attribute razreda koje služe kao brojači pristiglih GET, odnosno POST zahtjeva, kao i mapu *session* tipa *Map<Integer, MediaSession>* unutar koje se spremaju opisi sjednice višemedijskih usluga pristigli unutar tijela POST zahtjeva. Kao ključ zadane mape postavljen je atribut *postCounter* tipa *Integer*, koji predstavlja broj primljenog POST zahtjeva. Dodatno, razred *RequestHandler* sadrži metodu *requestBodyPrint(String requestBody)* koja kao ulazni parametar prima tijelo zahtjeva POST koje ispisuje u konzolu te *get* i *set* metode za dohvaćanje broja zahtjeva, tj. spremanje sjednice unutar definirane mape.

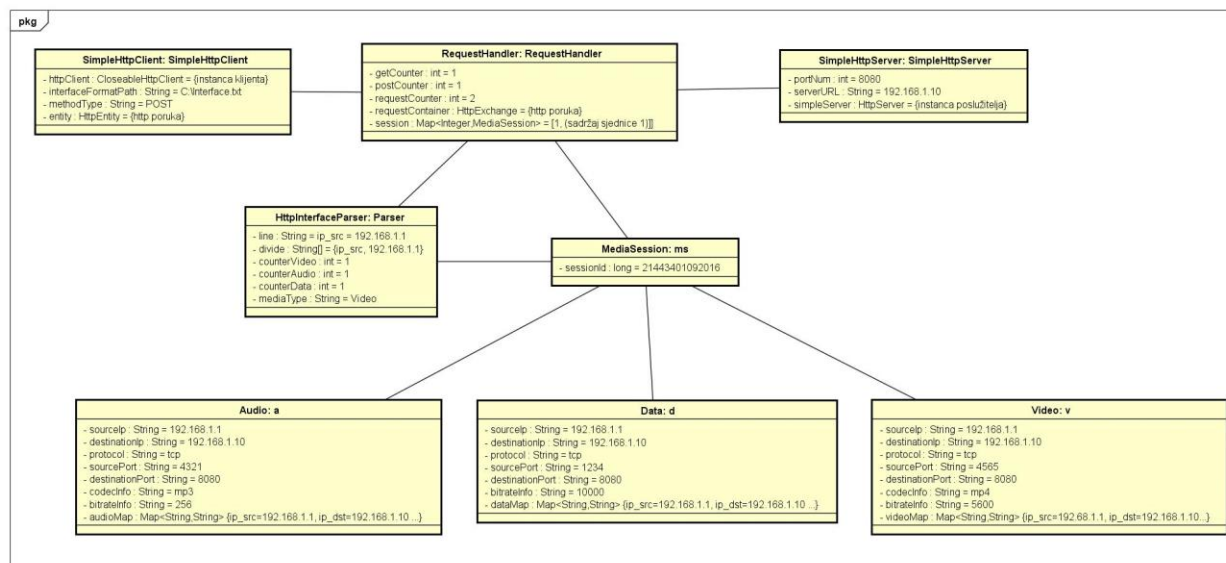
Nakon primljenog POST zahtjeva razred *RequestHandler* instancira objekt *parser* tipa *HttpInterfaceParser*, koji služi za obradu parametara sjednice višemedijske usluge u prethodno opisanom formatu. Razred *HttpInterfaceParser* sadrži nekoliko atributa koji služe isključivo kao „globalni“ spremnici podataka, kao što su trenutno pročitana linija teksta opisa višemedijske sjednice, broj video, audio i podatkovnih tokova medija unutar opisa sjednice, tip trenutno obrađivanog medija te spremnike za varijable tipa *Audio*, *Video*, *Data*, *MediaSession* koje sadrže podatke opisa pojedinog medija unutar sjednice. Također, razred *HttpInterfaceParser* sadrži tri metode *parseVideo*, *parseAudio*, *parseData* koje služe za obradu pojedine vrste medijskog toka

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

u opisu sjednice, a kao ulazne argumente primaju trenutni broj pojavljene vrste medija te para parametara koji predstavljaju npr. (ip_src, vrijednost). Osim toga, navedene metode sadrže referencu na metodu *matchParameters()* koja kao ulazne argumente prima tip medija te par trenutno obrađenih parametara. Prema skupu primljenih parametara metoda *matchParameters()* zapisuje podatke u za to predviđene attribute. Dodatno, metode *parseVideo*, *parseAudio* i *parseData* zapisuju opis svakog medija višemedijske sjednice u pojedine varijable tipa *Audio*, *Video*, *Data* i *MediaSession*. Važno je spomenuti kako razred *HttpInterfaceParser* sadrži metodu *parsing(String body)* koja kao ulazni parametar prima tijelo POST zahtjeva te ga obrađuje pozivajući gore navedene metode. Razred *HttpInterfaceParser* sadrži i metodu *getSession()* koja služi za dohvaćanje opisa sjednice, tj. varijable tipa *MediaSession*.

5.4.4 Dijagram objekata s opisom

Na *Slika 5-6* prikazan je HTTP API u jednom trenutku izvođenja. Prikaz je izrađen pomoću dijagrama objekata u programu *Astah Professional*. Kao što je vidljivo sa dijagrama, pojedina sjednica *MediaSession* se sastoji od jednog ili više medija tipa *Audio*, *Video* ili *Data*, od kojih svaki mediji karakteriziraju atributi prikazani na *Slika 5-3*.



Slika 5-6 Dijagram objekata HTTP API-ja

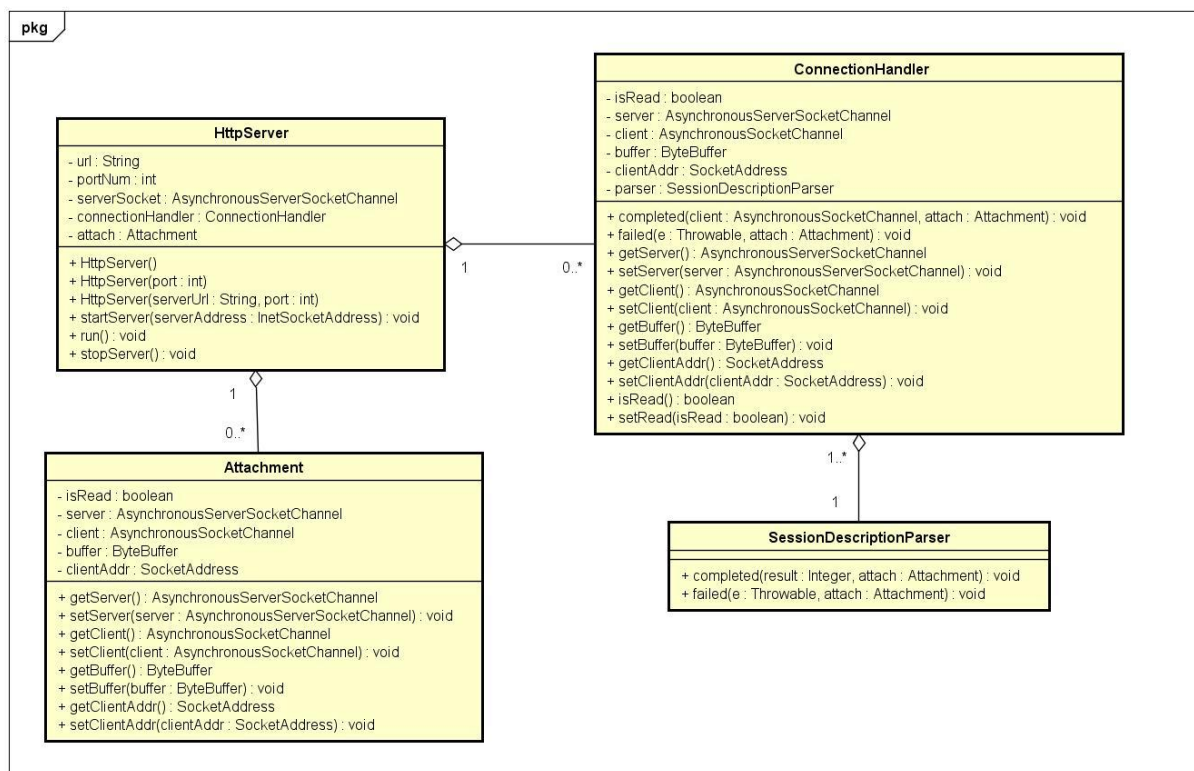
Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

5.4.5 Struktura programske izvedbe HTTP API-ja izrađenog korištenjem standardnih Java biblioteka

Za razliku od prvog HTTP API-ja koji koristi nestandardne knjižnice (*com.sun.net.httpserver*) sljedeći API koristi standardne Java knjižnice. Osim toga, ovaj API se izrađuje nižom razinom apstrakcije. U prevedenom značenju, ne postoje gotovi modeli (metode, razredi) za izradu poslužitelja ili klijenta. Struktura programske izvedbe API-ja se sastoji od jednog temeljnog dijela, tj. poslužitelja čija je funkcionalnost ostvarena pomoću razreda *HttpServer*. Zadaće prihvatanje veze i obrada poruke poslane od strane klijenta ostvaruju razredi *Attachement*, *ConnectionHandler*, *SessionDescriptionParser*. Detaljnije funkcionalnosti i sadržaj pojedinih razreda API-ja prikazane su kroz dijagram razreda u sljedećem poglavlju, napravljenim u jeziku UML.

5.4.5.1 Dijagram razreda s opisom

Na *Slika 5-7* prikazan je dijagram razreda HTTP API-ja izrađenog korištenjem standardnih Java biblioteke unutar *java.io* paketa.



Slika 5-7 Dijagram razreda HTTP API-ja izrađenog korištenjem standardnih Java knjižnica

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Razred *HttpServer* sastoji se od četiri atributa razreda koji predstavljaju IP adresu i vrata na kojima će poslužitelj osluškivati zahtjeve klijenata, priključnicu koja definira samog poslužitelja tipa *AsynchronousServerSocketChannel*, entiteta koji prihvaća veze od klijenta tipa *ConnectionHandler* te *placeholder*-a *Attachement* koji služi kao sučelje između entiteta za prihvatanje veze i entiteta za obradu zahtjeva klijenta. Također, razred se sastoji od 3 konstruktora izrađenih na temelju polimorfizma kako bi se omogućila maksimalna fleksibilnost te ostvarili prethodno navedeni funkcionalni zahtjevi. Osim toga, razred sadrži metodu *startServer* koja kao ulazni parametar prima adresu *serverAdress* tipa *InetSocketAddress* u obliku *IP adresa:vrata* te pokreće poslužitelj tj. postavlja priključnicu u stanje slušanja. Osim toga za ispravnu funkcionalnost poslužitelja pri njegovoj integraciji unutar izvornog koda upravljačkog uređaja, odnosno razreda *QoSRouting* potrebno je poslužitelj vezati uz vlastitu dretvu kako izvođenje koda poslužitelja ne bi blokiralo ostatak sustava. U tu svrhu izrađena je metoda *run* koja nadjačava istu metodu unutar razreda *Thread* kojeg razred *HttpServer* nasljeđuje. Također, razred sadrži metodu *stopServer* koja služi za zaustavljanje osluškivanja priključnice za funkcijom poslužitelja čime se oslobađaju zauzeta vrata i otpušta dretva uz koju je poslužitelj vezan.

Razred *ConnectionHandler* sastoji se od šest atributa razreda. Atribut *server* predstavlja instancu poslužitelja, dok atribut *client* tipa *AsynchronousSocketChannel* jasno, predstavlja instancu klijenta koji je uspostavio vezu sa poslužiteljem, odnosno poslao mu određeni zahtjev. Osim toga razred se sastoji od atributa *buffer* tipa *ByteBuffer* koji predstavlja spremnik poruke koju šalje klijent u obliku niza bajtova, atributa *isRead* tipa *Boolean* koji ima zadaću zastavice te drži informaciju je li poruka obrađena ili ne, kao i instancu entiteta *parser* koji služi za obradu pristigle poruke. Razred sadrži i metodu *completed* koja kao ulazne argumente prima instancu klijenta *client* te instancu razreda *Attachement* koji sadrži sve informacije potrebne za prihvatanje veze što je i glavna zadaća ove metode. Nakon ispravnog prihvata veze od strane klijenta i primljene poruke, zahtjev se proslijeđuje zadanom parseru na daljnju obradu. Ukoliko je došlo do pogreške u uspostavi veze poziva se metoda *failed* koja o tome obavještava korisnika.

Razred *Attachement* služi isključivo kao sučelje između razreda koji predstavlja poslužitelja *HttpServer* te razreda koji služi za prihvatanje veze od klijenta, *ConnectionHandler*. Prema tome, ovaj razred sadrži attribute koji su potrebni objema navedenim razredima a to su

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

spremnik poruke *buffer*, poveznicu na instancu poslužitelja *server*, poveznicu na instancu klijenta *client* te informaciju o obradi poruke tipa *boolean*. Osim toga kao što je vidljivo na *Slika 5-7* razred sadrži određene *get* i *set* metode koje služe za dohvat, odnosno pohranu navedenih atributa.

Razred *SessionDescriptionParser* ima zadaću prihvata poruke klijenta te njezinu daljnju obradu. Sastoji se od metode *completed* koja kao ulazni parametar prima atribut *result* tipa *Integer* koji predstavlja stanje obrade zahtjeva (stanje veze) te poveznicu na razred *Attachment* sa informacijama potrebnim o poruci. Također, razred sadrži metodu *failed* koja se poziva u slučaju pogreške pri obradi poruke.

Na *Slika 5-8* prikazan je isječak koji predstavlja dio koda gdje poslužitelj prihvata vezu zatraženu od klijenta te primljenu poruku prosljeđuje parseru na daljnju obradu.

```
@Override
public void completed(AsynchronousSocketChannel client, Attachment attach) {
    try {
        SocketAddress clientAddr = client.getRemoteAddress();
        System.out.println("Accepted a connection from " + clientAddr);
        attach.getServer().accept(attach, this);
        parser = new SessionDescriptionParser();
        Attachment newAttach = new Attachment();
        newAttach.setServer(attach.getServer());
        newAttach.setClient(client);
        newAttach.setBuffer(ByteBuffer.allocate(2048));
        newAttach.isRead = true;
        newAttach.setClientAddr(clientAddr);
        client.read(newAttach.getBuffer(), newAttach, parser);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
public void failed(Throwable e, Attachment attach) {
    System.out.println("Failed to accept a connection.");
    e.printStackTrace();
}
```

Slika 5-8 Isječak koda prihvata veze klijenta i prosljeđivanje poruke parseru

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

5.4.6 Primjer izvođenja API-ja

U svrhu testiranja i prezentacije funkcionalnosti implementiranog API-ja koristi se unaprijed definirani opis sjednice višemedijskih usluga u obliku tekstualnog dokumenta pod nazivom *SessionFormat.txt* prikazanog na *Slika 5-3*.

Prilikom pokretanja API-ja dočeka nas poruka u obliku prikazanom na *Slika 5-9*. Unosom 0, 1 ili 2 u konzoli omogućava se postavljanje parametara poslužitelja prema navedenim funkcijskim zahtjevima.

```
Launcher (1) [Java Application] C:\Program Files\Java\jdk1.8.0_60\bin\javaw.exe (Jan 11, 2016, 9:13:26 PM)
Start server: DEFAULT (0) / CUSTOM PORT (1) / CUSTOM PORT & ADDRESS (2)
```

Slika 5-9 Primjer izvođenja HTTP API-ja: odabir načina pokretanja poslužitelja

U sljedećim primjera prikazan je odabir 1, odnosno pokretanje poslužitelja na *localhost* adresi računala sa korisnički definiranim vratima. Nakon odabira *CUSTOM PORT*, odnosno unosa 1 u konzolu, od korisnika se traži unos vrata na kojima će poslužitelj biti pokrenut, uz informaciju mogućeg raspona vrijednosti vrata. Unosom broja ulaznih vrata koja su slobodna pokreću se HTTP klijent i HTTP poslužitelj te ispisuje povratna poruka o uspješno pokrenutom poslužitelju, klijentu te adresi i statusu poslužitelja. Nakon toga korisnik na izbor dobiva slanje POST odnosno GET zahtjeva (*Slika 5-10*).

```
Start server: DEFAULT (0) / CUSTOM PORT (1) / CUSTOM PORT & ADDRESS (2)
1
Enter port number(0-65535):
5555
Server started successfully!
Server status: ONLINE
Server address: localhost:5555
HTTP client created

==== Options ====
(0) Send GET request
(1) Send POST request
(2) Send File over POST request
```

Slika 5-10 Primjer izvođenja HTTP API-ja: pokretanje poslužitelja i klijenta

Odabirom opcije (1) *Send POST request* HTTP klijent čita podatke o višemedijskoj

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

sjednici, postavlja ih u tijelo POST zahtjeva te šalje poslužitelju. U trenutku kada poslužitelj primi POST zahtjev on se prosljeđuje na obradu a informacije o zahtjevu ispisuju na ekran (*Slika 5-11*).

```
Request is being processed by server...
Request processed!
Request number: 1

===== REQUEST MESSAGE =====
POST /get HTTP/1.1
Date: Sun, 24 Jan 2016 16:19:20 GMT
Host: 127.0.0.1
Request body:
sessionId=17565501092016
m=video
ip_src=192.168.1.1
ip_dst=192.168.1.31
proto=udp
port_src=12345
port_dst=54321
codec=mp4
bitrate=5600
```

Slika 5-11 Primjer izvođenja HTTP API-ja: primljeni POST zahtjev

Poslužitelj obrađuje zahtjev koristeći razrede i metode navedene u poglavlju 4.4.3, prema potrebne podatke o sjednici te prilikom uspješne obrade šalje povratnu poruku klijentu (čiji je format prikazan na *Slika 5-12*). Nakon primljene poruke prekida se rad HTTP API-ja zaustavljanjem poslužitelja i klijenta.

```
===== RESPONSE MESSAGE=====
Status line: HTTP/1.1 200 OK
Method: POST
Date: Sun, 24 Jan 2016 15:19:20 GMT
Content-type: text/plain
Content-length: 2
...Response Body...
OK
===== END =====
Server closed!
```

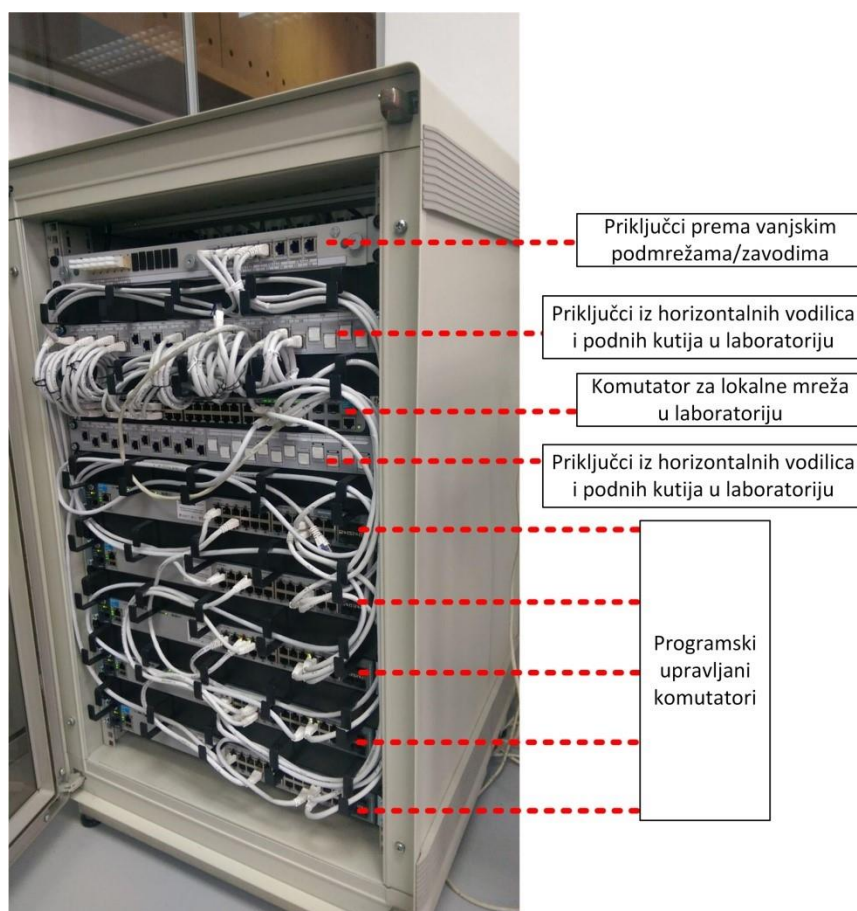
Slika 5-12 Primjer izvođenja HTTP API-ja: povratna poruka uspješno obrađenog POST zahtjeva

Primjeri izvođenja drugog API-ja prikazani su sljedećem poglavlju, gdje je integriran unutar upravljačkog uređaja OpenDaylight.

Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

6. Demonstracija rada upravljačkog uređaja OpenDaylight

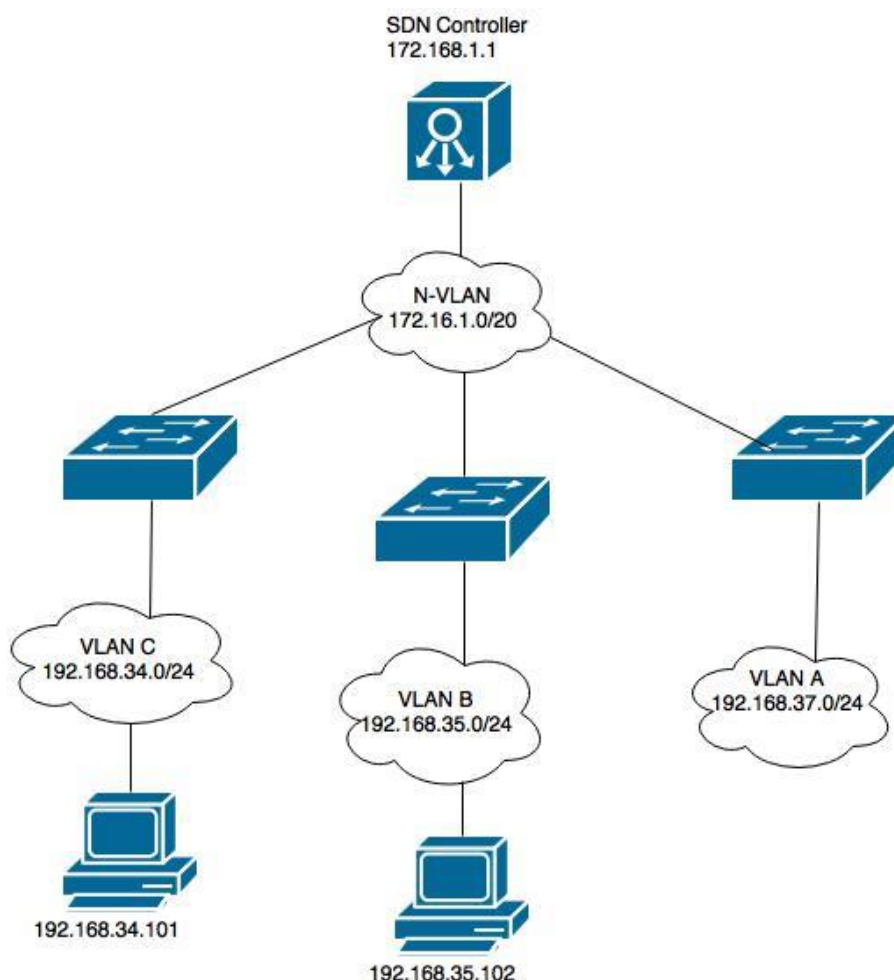
Komunikacijska mreža unutar laboratorija Zavoda za telekomunikacije sastoji se od mrežnog ormara unutar kojeg se nalaze četiri *HP 2920-24G* komutatora koji podržavaju *OpenFlow* standard te računala sa Linux i Windows operacijskim sustavima. Mrežni ormar koji sadrži navedene komutatore prikazan je na *Slika 6-1*.



Slika 6-1 Raspored komunikacijske opreme u mrežnom ormaru

Logička topologija mreže unutar ovoga projekta sastoji se od *OpenDaylight Hydrogen* upravljačkog uređaja, tri *OpenFlow* komutatora gdje se svaki uređaj nalazi u posebnom VLAN-u (N-VLAN, VLAN-A, VLAN-B, VLAN-C). Također, topologiju čine i dva krajnja uređaja (računala) koja su spojena na komutatore koji pripadaju VLAN-A, odnosno VLAN-B. Struktura logičke topologije prikazana je na *Slika 6-2*.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.



Slika 6-2 Logička topologija projektne SDN mreže

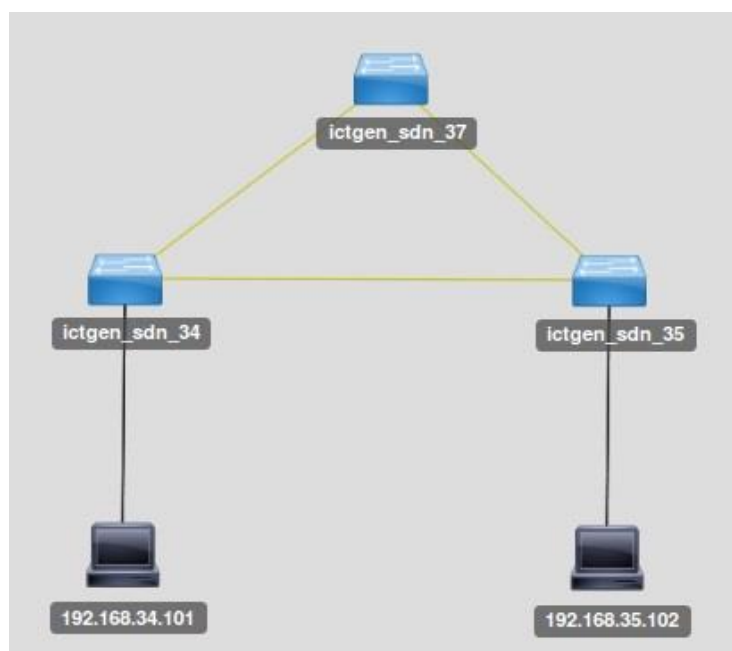
Nakon proširenja izvornog koda usmjeravanja unaprijed su definirani tokovi namijenjeni DHCP zahtjevima i odgovorima te je omogućeno dinamičko dodavanje krajnjih uređaja unutar logičke topologije upravljačkog uređaja *OpenDaylight* što se inicira slanjem ARP zahtjeva kao što je prikazano na *Slika 6-3*.

```
Installing predefined flow for all nodes!
Setting node OF|00:02:d0:bf:9c:d6:10:80 so as to send packets [any, 255.255.255.255, udp, any, 67] to the local hardware path (non-OpenFlow) for processing
Installing new flow on: OF|00:02:d0:bf:9c:d6:10:80 OF|1 ... ok
Setting node OF|00:02:d0:bf:9c:d6:10:80 so as to send packets [any, 255.255.255.255, udp, any, 68] to the local hardware path (non-OpenFlow) for processing
Added host: 192.168.34.101, d0:50:99:37:79:a1, 2
Added host: 192.168.35.102, d0:50:99:37:77:7b, 2
```

Slika 6-3 Povratna poruka o dodanom krajnjem uređaju (engl. host)

Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Dodavanjem krajnjih uređaja unutar logičke topologije omogućena je njihova direktna komunikacija. Određivanje puta kojim se komunikacija odvija računa se dinamički uporabom algoritma usmjeravanja zasnovanog na optimizaciji kolonijom mrava. Funkcionalnost je testirana naredbom *ping*. Ispravnost implementiranih proširenja vidljiva je kroz GUI upravljačkog uređaja OpenDaylight kao što je vidljivo na *Slika 6-4*.



Slika 6-4 Topologija mreže unutar GUI-a upravljačkog uređaja OpenDaylight

Kako bi se postigao krajnji cilj projekta unutar upravljačkog uređaja implementiran je jednostavan web poslužitelj koji se pokreće zajedno sa upravljačkim uređajem te se izvodi u njegovoj pozadini. Integrirani web server predstavlja HTTP API izrađen pomoću standardnih Java knjižnica koji detaljnije opisan u poglavlju 5.4.5. Primjeri njegovog izvođenja, tj. pokretanja i obrade HTTP GET zahtjeva prikazani su na *Slika 6-5*, odnosno *Slika 6-6*.

```

Starting server....
Removing all flows for all nodes!
===== Server started =====
Server is listening at: 172.16.1.1:32000
  
```

Slika 6-5 Pokretanje integriranog web poslužitelja

Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```
osgi> Accepted a connection from /172.16.1.1:46424
===== NEW REQUEST =====
Client: /172.16.1.1:46424
GET / HTTP/1.1
Host: 172.16.1.1:32000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Slika 6-6 Primitak HTTP GET zahtjeva

Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

7. Zaključak

SDN mreže pružaju mogućnosti ubrzanog razvoja temeljenog na razdvajanju upravljanja mrežom od prosljeđivanja podataka, a te su mogućnosti podržane od strane upravljačkog uređaja OpenDaylight i OpenFlow protokola. Implementirajući razne algoritme usmjeravanja u SDN mrežama, u našem slučaju algoritma usmjeravanja zasnovanog na optimizaciji kolonijom mrava, mogu se poboljšati performanse mreža, kao što su kašnjenje i propusnost.

Cilj ovog projekta bilo je upoznavanje s konceptom i funkcionalnošću SDN mreža, uključujući rad na proširenjima algoritma usmjeravanja ACO, njegove implementacije unutar laboratorijske mreže te izrade HTTP sučelja potrebnog za spajanje izvedene mreže s aplikacijom Showroom koja je bila zadatak drugog projektnog tima.

Unutar postojećeg proširenja za OpenDaylight upravljački uređaj, QoSrouting, ugrađene su sljedeće dodatne funkcionalnosti: mogućnost obrađivanja paketa s VLAN zaglavljima, dodavanje informacija o krajnjim uređajima, metoda za postavljanje ranije definiranih tokova i HTTP server.

Prijedlozi da daljnji rad uključuju: povezivanje upravljačkog uređaja s drugim višemedijskim aplikacijama (npr. aplikacija *Showroom*) putem implementiranog HTTP API-ja, te prijenos cjelokupnog proširenja u noviju verziju upravljačkog uređaja (npr. OpenDaylight Lithium) ili implementirati procjenu gubitka na poveznicama mrežnih uređaja za ACO algoritam uz pomoć koda iz dodatka B ovog dokumenta (str. 87).

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

8. Literatura

- [1] Matija Šantl, *Usmjeravanje u programski upravljanoj mreži zasnovano na parametrima kvalitete višemedijske usluge*, diplomski rad, Zagreb 2015.
- [2] Karlo Božić, *Model aplikacijskog programskog sučelja za upravljanje kvalitetom višemedijskih usluga u programski upravljanoj mreži*, završni rad, Zagreb 2015.
- [3] N. McKeown, T. Andreson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, *OpenFlow: Enabling Innovation in Campus Networks*, ACM SIGCOMM Computer Communication Review, Volume 38, SAD 2008.
- [4] Open Network Foundation, *OpenFlow Switch Specification version 1.0.0*, 2009.
- [5] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, *Software-Defined Networking: A Comprehensive Survey*, 2014.
- [6] Ole. J. Jacobsen, Software-Defined Networks and OpenFlow, *The Internet Protocol Journal*, volume 16, number 1, 2013.
- [7] OpenDaylight Community, *OpenDaylight Getting Started Guide*, Lithium, 2015.
- [8] OpenDaylight Community, *OpenDaylight Developer Guide*, Lithium, 2015.
- [9] Marko Čupić, *Programiranje u Javi*, verzija 0.3.26, Zagreb 2015.
- [10] Allen B. Downey, *Think Java, How to Think Like a Computer Scientist*, 2012.
- [11] Mkyoung, *RESTful Java client with Apache HttpClient*, <http://www.mkyong.com/webservices/jax-rs/restful-java-client-with-apache-httpclient/>, 27.12.2015.
- [12] Mkyoung, *Apache HttpClient Examples*, <http://www.mkyong.com/java/apache-httpclient-examples/>, 27.12.2015.
- [13] Mkyoung, *How to send HTTP request GET/POST in Java*, <http://www.mkyong.com/java/how-to-send-http-request-getpost-in-java/>, 27.12.2015.
- [14] HTTP server, *Have a simple HTTP server*, <http://www.rgagnon.com/javadetails/java-have-a-simple-http-server.html>, 27.12.2015.
- [15] Andry Feng, *Create a Simple Web Server in Java (1) - HTTP Server*, 15.10.2015., <http://www.codeproject.com/Tips/1040097/Create-a-Simple-Web-Server-in-Java-HTTP-Server>, 27.12.2015
- [16] O. Kalnichevski, J. Moore, J. van Grup, *HttpClient Tutorial*, dostupno na <http://hc.apache.org/>
- [17] O. Kalnichevski, *HttpCore Tutorial*, dostupno na <http://hc.apache.org/>
- [18] Developer Cisco, *Cisco XNC 1.5.0 API*, dostupno na <https://developer.cisco.com/media/XNCJavaDocs/overview-summary.html>

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

- [19] S. G. Yaseen, N. M. A. Al-Slamy, Ant Colony Optimization, International Journal of Computer Science and Network Security, VOL.8 No.6, June 2008, dostupno na http://paper.ijcsns.org/07_book/200806/20080649.pdf
- [20] OpenDaylight platforma, dostupno na službenim stranicama <https://www.opendaylight.org/>
- [21] OpenDaylight Delivers Open Source Software to Enable Software-Defined Networking, <https://www.opendaylight.org/news/platform-news/2014/02/opendaylight-delivers-open-source-software-enable-software-defined>, 19.01.2016.
- [22] GettingStarted:Pulling, Hacking, and Pushing All the Code from the CLI, [https://wiki.opendaylight.org/view/GettingStarted:Pulling, Hacking, and Pushing All the Code from the CLI](https://wiki.opendaylight.org/view/GettingStarted:Pulling,_Hacking,_and_Pushing_All_the_Code_from_the_CLI), 22.01.2016.
- [23] Open Networking Foundation, *Principles And Practices for Securing Software-Defined Networks*, Version 1.0, January 2015.

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

9. Skraćenice

HTML	<i>HyperText Markup Language</i>
IP	<i>Internet Protocol</i>
SDN	<i>Software-Defined Network</i>
HTTP	<i>HyperText Transfer Protocol</i>
IPTV	<i>Internet Protocol Television</i>
QoS	<i>Quality of Service</i>
ONF	<i>Open Network Foundation</i>
NOS	<i>Network Operating System</i>
A-CPI	<i>Application Plane Interface</i>
NBI	<i>Northbound Interface</i>
TLS	<i>Transport Layer Security</i>
SSL	<i>Secure Sockets Layer</i>
ACO	<i>Ant Colony Optimization</i>
MTU	<i>Maximum Transmission Unit</i>
API	<i>Application Programming Interface</i>
REST	<i>Representational State Transfer</i>
WWW	<i>World Wide Web</i>
UML	<i>Unified Modeling Language</i>
OSGi	<i>Open Service Gateway initiative</i>
SAL	<i>Service Abstraction Layer</i>
OVSDB	<i>Open vSwitch Database</i>
AD-SAL	<i>API driven SAL</i>
MD-SAL	<i>Model driven SAL</i>
MAC	<i>Media Access Control</i>
VLAN	<i>Virtual Local Area Network</i>
ARP	<i>Address Resolution Protocol</i>
ID	<i>Identification</i>
L2	<i>Layer 2</i>
APT	<i>Advanced Package Tool</i>

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

UI

User Interface

DLUX

OpenDaylight User Experience

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

10. Rječnik pojmova

SDN (engl. Software-Defined Networking) - Programski upravljana mreža

OpenDaylight - otvorena programska platforma za realizaciju SDN upravljačkog uređaja (uređaj otvorenog koda napisan u programskom jeziku Java)

OpenFlow - otvoreni standard koji omogućuje komunikaciju upravljačkog uređaja i mrežnih uređaja u SDN mreži

Hydrogen, Lithium - OpenDaylight distribucije

HTTP (engl. Hypertext Transfer Protocol) - platformski neovisan aplikacijski protokol, temeljen na modelu klijent-poslužitelj

Poslužitelj - program koji dostavlja uslugu drugim programima koji su spojeni na njega preko komunikacijskog kanala

Klijent - program koji pristupa poslužitelju (ili više njih) tražeći uslugu

Klijent-poslužitelj - vrsta arhitekture programske podrške

Java - objektno-orijentirani programski jezik visoke razine

Eclipse - otvorena platforma (alat) bazirana na Java tehnologiji namijenjena razvoju aplikacija

CLI (engl. command-line interface) - tekstualno naredbeno sučelje

Mininet - mrežni emulator sa CLI sučeljem

ACO (engl. Ant Colony Optimization) - optimizacija kolonijom mrava

MTU (engl. Maximum Transmission Unit) - najveća transportna jedinica u mreži.

Biblioteka (engl. Library) - skup resursa koji korišteni od strane računalnih programa pri njihovom razvoju. Oni mogu uključivati konfiguracijske datoteke, dokumentaciju, unaprijed napisan kod, rutine, razrede, vrijednosti ili tipove specifikacija.

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Popis slika

Slika 1-1 Studijski slučaj: integracija višemedijskih aplikacija i uređaja OpenDaylight	6
Slika 2-1 Arhitektura programski upravljane komunikacijske mreže	8
Slika 3-1 Arhitektura platforme OpenDaylight, verzija Hydrogen	16
Slika 3-2 Shema upravljačkog uređaja unutar platforme OpenDaylight	17
Slika 3-3 Najvažniji korišteni paketi, razredi, sučelja i metode	25
Slika 3-4 Pokretanje upravljačkog uređaja OpenDaylight, verzija Lithium	33
Slika 3-5 Obrazac za prijavu na grafičko sučelje platforme OpenDaylight, verzija Lithium.....	34
Slika 4-1 Dijagram razreda QoSRouting s metodom za izračun gubitaka na poveznici	37
Slika 4-2 Dijagram razreda AntColony s mapom za pohranu gubitaka paketa na poveznicama .	38
Slika 4-3 Isječak programskog koda za izračun gubitka na poveznicama.....	39
Slika 4-4 Kretanje marava u potrazi za hranom.....	40
Slika 5-1 Model klijent-poslužitelj	44
Slika 5-2 Primjer razmjene HTTP poruka	46
Slika 5-3 Format opisa sjednice višemedijske usluge.....	48
Slika 5-4 Dijagram razreda modeliranog API-ja	50
Slika 5-5 Programski isječak instanciranja HTTP poslužitelja i funkcije za prijem zahtjeva (RequestHandler)	52
Slika 5-6 Dijagram objekata HTTP API-ja.....	53
Slika 5-7 Dijagram razreda HTTP API-ja izrađenog korištenjem standardnih Java knjižnica	54
Slika 5-8 Isječak koda prihvata veze klijenta i proslijeđivanje poruke parseru	56
Slika 5-9 Primjer izvođenja HTTP API-ja: odabir načina pokretanja poslužitelja.....	57
Slika 5-10 Primjer izvođenja HTTP API-ja: pokretanje poslužitelja i klijenta	57
Slika 5-11 Primjer izvođenja HTTP API-ja: primljeni POST zahtjev.....	58
Slika 5-12 Primjer izvođenja HTTP API-ja: povratna poruka uspješno obrađenog POST zahtjeva	58
Slika 6-1 Raspored komunikacijske opreme u mrežnom ormaru	59
Slika 6-2 Logička topologija projektne SDN mreže	60
Slika 6-3 Povratna poruka o dodanom krajnjem uređaju (engl. host)	60
Slika 6-4 Topologija mreže unutar GUI-a upravljačkog uređaja OpenDaylight	61

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Slika 6-5 Pokretanje integriranog web poslužitelja	61
Slika 6-6 Primitak HTTP GET zahtjeva	62

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Dodatak A - Programska izvedba HTTP API-ja

Razred *SimpleHttpServer*

```

package com.unizg.fer.tel.sdn.opendaylight.httpinterface;

import java.io.IOException;
import java.net.BindException;
import java.net.InetAddress;
import java.net.InetSocketAddress;

import com.sun.net.httpserver.HttpServer;

/*
 * Class which represents simple HTTP server
 * @author Vedran Serenčeš
 */
public class SimpleHttpServer {
    private HttpServer simpleServer;
    private int portNum;
    private String serverURL;
    private RequestHandler requestHandler; // the purpose of RequestHandler is
                                           // to process
    HTTP request
    private boolean serverStatus;

    /*
     * Creating an instance of HTTP server object on default address and port
     * (localhost, 5555)
     */

    public SimpleHttpServer() {
        this.portNum = 5555;
        this.serverURL = InetAddress.getLoopbackAddress().getHostName();
        this.requestHandler = new RequestHandler();
        startServer();
    }

    /*
     * Creating an instance of HTTP server object on default address and user
     * configured port
     */
    public SimpleHttpServer(int port) {
        this.portNum = port;
        this.serverURL = InetAddress.getLoopbackAddress().getHostName();
        this.requestHandler = new RequestHandler();
        startServer();
    }

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

    }

    /*
    * Create an instance of HTTP server object on user configured address and
    * port
    */

    public SimpleHttpServer(InetAddress ipAddress, int port) {
        this.portNum = port;
        this.serverURL = ipAddress.getHostAddress();
        this.requestHandler = new RequestHandler();
        startServer();
    }

    /*
    * Starting server and printing result message
    */
    public void startServer() {
        try {
            this.simpleServer = HttpServer.create(new
InetSocketAddress(serverURL, portNum), 0);
        } catch (BindException e) {
            System.out.println("ERROR: Server cannot bind to selected port and IP
address!");
            System.out.println("ERROR! Server couldn't be created.");
            e.printStackTrace();
        } catch (NullPointerException e) {
            System.out.println("ERROR: IP address (IP address not set/wrong
format) !");
            System.out.println("ERROR! Server couldn't be created.");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("ERROR! Server couldn't be created.");
            e.printStackTrace();
        }
    }

    /*
    * HttpContext represents a mapping between the root URI path of a web
    * service to a HttpHandler which is invoked to handle requests destined
    * for that path on the associated container.
    */
    this.simpleServer.createContext("/get", requestHandler);
    this.simpleServer.setExecutor(null);
    this.simpleServer.start();
    this.serverStatus = true;
    System.out.println("Server started successfully!");
    getServerInfo();
}

/*
* Printing server status informations

```


Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

    */
    public void getServerInfo() {
        if (serverStatus == true) {
            System.out.println("Server status: ONLINE");
            System.out.println("Server address: " + serverURL + ":" + portNum);
        } else {
            System.out.println("Server status: OFFLINE");
        }
    }

    /*
    * Stopping Http Server
    */
    public void stopServer() {
        simpleServer.stop(0);
    }

    /*
    * Getter for server running (listening) port
    */
    public int getPort() {
        return this.portNum;
    }

    /*
    * Getter for server IP adress
    */
    public String getUrl() {
        return this.serverURL;
    }

    /*
    * Getter for Request Handler
    */
    public RequestHandler getRequestHandler() {
        return this.requestHandler;
    }

    /*
    * Getter for server status, true=online, false=offline
    */
    public boolean getServerStatus() {
        return this.serverStatus;
    }
}

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Razred *SimpleHttpClient*

```
package com.unizg.fer.tel.sdn.opendaylight.httpinterface;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.entity.mime.MultipartEntityBuilder;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;

/*
 * Class which represents simple HTTP client
 * @author: Vedran Serenčes
 */
public class SimpleHttpClient {
    private CloseableHttpClient httpClient;
    private String sessionFormatPath;
    private HttpEntity entity;

    /**
     * Creating an instance of HTTP client
     */
    public SimpleHttpClient() throws IOException {
        try {
            setHttpClient(HttpClients.createDefault());
            System.out.println("HTTP client created");
        } catch (Exception e) {
            System.out.println("ERROR: HTTP client not created!");
        }
    }

    /**
     * Sending GET request to server
     */
    public void sendHttpGet(String url) throws IOException, ClientProtocolException {
        HttpGet httpGet = new HttpGet(url);
```

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

        System.out.println("Sending GET request to server...");
        try {
            CloseableHttpResponse response = httpClient.execute(httpGet);
            processResponse(response);
        } catch (IOException e) {
            System.out.println("ERROR: Connection Aborted / HTTP protocol error");
        }
    }

    /**
     * Sending POST request to server
     *
     * @param url the server
     */
    public void sendHttpPost(String url) throws IOException, ClientProtocolException {
        HttpPost httpPost = new HttpPost(url);
        setRequestMessage(httpPost);
        System.out.println("Sending POST request to server...");
        try {
            CloseableHttpResponse response = httpClient.execute(httpPost);
            // Function for response processing
            processResponse(response);
        } catch (IOException e) {
            System.out.println("ERROR: Connection Aborted / HTTP protocol error");
        }
    }

    /**
     * Sending txt file containing parameters of multimedia session via POST
     * request
     */
    public void sendFilePost(String url) throws ClientProtocolException, IOException {
        HttpPost httpPost = new HttpPost(url);
        MultipartEntityBuilder builder = MultipartEntityBuilder.create();
        File sessionFile = new File(sessionFormatPath);
        builder.addBinaryBody("MultimediaSession", sessionFile);
        httpPost.setEntity(builder.build());
        CloseableHttpResponse response = httpClient.execute(httpPost);
        processResponse(response);
    }

    /**
     * Processing response message
     */
    private void processResponse(CloseableHttpResponse response) throws IOException {
        try {
            this.entity = response.getEntity();// respond from server
            String responseBody = EntityUtils.toString(entity);
            Header[] responseHeader = response.getAllHeaders();

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

        String statusLine = response.getStatusLine().toString();
        EntityUtils.consume(entity);
        /*
         * Printing response message informations
         */
        System.out.println("\n===== RESPONSE MESSAGE=====");
        System.out.println("Status line: " + statusLine);
        System.out.println(responseHeader[0]);
        System.out.println(responseHeader[1]);
        System.out.println(responseHeader[2]);
        System.out.println(responseHeader[3]);
        System.out.println("...Response Body...");
        System.out.println(responseBody);
        System.out.println("===== END =====");
    }

    catch (Exception e) {
        System.out.println("ERROR: Failed to process response!");
    } finally {
        response.close();// close connection with server
        httpClient.close();
    }
}

/*
 * Setting content of message body (content of Interface.txt) for POST
 * request in the form of String
 *
 * @param httpPost the POST request
 */
public void setRequestMessage(HttpPost httpPost) throws IOException {
    BufferedReader br = new BufferedReader(new
FileReader(this.sessionFormatPath));
    try {
        StringBuilder sb = new StringBuilder();
        String line = br.readLine();

        while (line != null) {
            sb.append(line);
            sb.append(System.lineSeparator());
            line = br.readLine();
        }
        String everything = sb.toString();
        StringEntity input = new StringEntity(everything);
        httpPost.setEntity(input);
    } finally {
        br.close();
    }
}

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

    /*
     * Getter for HTTP client object
     */
    public HttpClient getHttpClient() {
        return httpClient;
    }

    /*
     * Setter for HTTP client object
     */
    public void setHttpClient(CloseableHttpClient httpClient) {
        this.httpClient = httpClient;
    }

    /*
     * Setter for path of txt file containing parameters of multimedia session
     */
    public void setSessionFormatPath(String path) {
        this.sessionFormatPath = path;
    }

    /*
     * Getter for path of txt file containing parameters of multimedia session
     */
    public String getSessionFormatPath() {
        return this.sessionFormatPath;
    }

    /*
     * Getter for an entity that can be sent or received with an HTTP message
     */
    public HttpEntity getEntity() {
        return this.entity;
    }
}

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Razred *RequestHandler*

```
package com.unizg.fer.tel.sdn.opendaylight.httpinterface;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

import com.sun.net.httpserver.Headers;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.unizg.fer.tel.sdn.opendaylight.http.parser.HttpSessionParser;
import com.unizg.fer.tel.sdn.opendaylight.http.parser.MediaSession;

/*
 * Class which processes HTTP request
 * @author Vedran Serenčeš
 */

public class RequestHandler implements HttpHandler {
    private Map<Integer, MediaSession> session = new HashMap<Integer,
MediaSession>();
    private int getCounter;
    private int postCounter;
    private int requestCounter;
    private HttpExchange requestContainer;
    private HttpSessionParser parser;

    /*
     * Main method for handling received request
     *
     * @param exchange the request placeholder
     */
    public void handle(HttpExchange exchange) throws IOException {
        System.out.println("Request recieved!");
        this.requestContainer = exchange;
        this.requestCounter++;
        String body = requestProcess();

        requestMessagePrint(body);

        if (body.equals(null) || !body.equals("")) {
            parser = new HttpSessionParser();
        }
    }
}
```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

        parser.parsing(body);
        this.postCounter++;
        this.session.put(postCounter, parser.getSession());
    } else {
        this.getCounter++;
        requestMessagePrint(body);
    }
    responseBuild();
}

/*
 * Processing body of POST request
 *
 * @Converting request from InputStream type to String
 */
public String requestProcess() throws IOException {
    System.out.println("\nRequest is being processed by server...");
    InputStreamReader input = new
InputStreamReader(this.requestContainer.getRequestBody(), "utf-8");
    BufferedReader br = new BufferedReader(input);
    StringBuilder sb = new StringBuilder();
    String body;
    while ((body = br.readLine()) != null) {
        sb.append(body + "\n");
    }
    body = sb.toString();
    return body;
}

/*
 * Building response message
 */
public void responseBuild() throws IOException {
    // Setting date format
    SimpleDateFormat timeFormat = new SimpleDateFormat("EEE, d MMM yyyy
HH:mm:ss 'GMT'");
    Date responseTime = new Date();
    /*
     * Setting response message, headers and body
     *
     * @Headers: Date, Content-Type, Method, Body lenght
     *
     * @Response Body = OK
     */
    Headers responseHeaders = this.requestContainer.getResponseHeaders();
    responseHeaders.set("Date", timeFormat.format(responseTime).toString());
    responseHeaders.set("Content-Type", "text/plain");
    responseHeaders.set("Method", this.requestContainer.getRequestMethod());
    String response = "OK";
    this.requestContainer.sendResponseHeaders(200, response.length());
}

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

        OutputStream os = this.requestContainer.getResponseBody();
        os.write(response.getBytes());
        os.close();
    }

    /**
     * Print body content of request message
     *
     * @param requestBody the body of recieved request in form of String
     */
    public void requestMessagePrint(String requestBody) throws IOException {
        /**
         * If recieved request == GET request or POST without body content set
         * body = "" -> style purpose
         */
        System.out.println("Request processed!");
        if (requestBody.equals("") || requestBody.equals(null)) {
            requestBody = "";
        }
        SimpleDateFormat timeFormat = new SimpleDateFormat("EEE, d MMM yyyy
HH:mm:ss 'GMT'");
        Date requestTime = new Date();
        System.out.println("Request number: " + this.requestCounter);
        System.out.println("\n===== REQUEST MESSAGE =====");
        System.out.println(this.requestContainer.getRequestMethod() + " " +
this.requestContainer.getRequestURI() + " "
+ this.requestContainer.getProtocol());
        System.out.println("Date: " + timeFormat.format(requestTime).toString());
        System.out.println("Host: " +
this.requestContainer.getLocalAddress().getHostName());
        System.out.println("Request body:\n" + requestBody);
        System.out.println("==== END ====");
    }

    /**
     * Getter for request counter
     */
    public int getRequestCounter() {
        return this.requestCounter;
    }

    /**
     * Getter for GET request counter
     */
    public int getGetCounter() {
        return this.getCounter;
    }

    /**
     * Getter for POST request counter

```


Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

    */
    public int getPostCounter() {
        return this.postCounter;
    }

    /*
     * Getter for request placeholder
     */
    public HttpExchange getRequestContainer() {
        return this.requestContainer;
    }

    /*
     * Getter for object type HttpInterfaceParser
     */
    public HttpSessionParser getParser() {
        return this.parser;
    }

    /*
     * Getter for map containing Multimedia Sessions
     */
    public Map<Integer, MediaSession> getSession() {
        return this.session;
    }
}

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Razred *HttpSessionParser*

```
package com.unizg.fer.tel.sdn.opendaylight.http.parser;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.StringReader;

/*
 * Class HttpSessionParser parses the format of multimedia session
 * @Author Vedran Serenčes
 */
public class HttpSessionParser {
    private String line = null;
    private String[] divide;
    private int counterVideo = 0;
    private int counterAudio = 0;
    private int counterData = 0;
    private String mediaType;
    private MediaSession session = new MediaSession();
    private Video v;
    private Audio a;
    private Data d;

    boolean flag = true; // successfull parsing -> for future
                        // implementations

    /*
     * Main method for parsing body of POST request
     *
     * @param body the body of POST request message in form of String
     */
    public void parsing(String body) {
        BufferedReader reader = new BufferedReader(new StringReader(body));
        try {
            try {
                line = reader.readLine();
                if (line.startsWith("-")) {
                    for (int i = 0; i < 4; i++) {
                        line = reader.readLine();
                    }
                    line = reader.readLine();
                }
            } catch (IOException e) {
                System.out.println("ERROR: Parsing request failed!");
                e.printStackTrace();
            }
            divide = line.split("=");
        }
    }
}
```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

        session.setSessionId(Long.parseLong(divide[1].trim()));
    /*
     * Main logic of parser
     */
    while ((line = reader.readLine()) != null && !line.contains("--")) {
        divide = line.split("=");
        if (divide[0].trim().equals("m")) {
            if (divide[1].trim().equals("video")) {
                counterVideo++;
                this.v = new Video();
                this.mediaType = "v";
            } else if (divide[1].trim().equals("audio")) {
                counterAudio++;
                this.a = new Audio();
                this.mediaType = "a";
            } else if (divide[1].trim().equals("data")) {
                counterData++;
                this.d = new Data();
                this.mediaType = "d";
            }
        } else {
            if (this.mediaType.equals("v")) {
                parseVideo(counterVideo, divide);
            } else if (this.mediaType.equals("a")) {
                parseAudio(counterAudio, divide);
            } else if (this.mediaType.equals("d")) {
                parseData(counterData, divide);
            }
        }
    }

    } catch (IOException e) {
        this.flag = false;
    } finally {
        try {
            reader.close();
        } catch (IOException e) {
            this.flag = false;
        }
    }
}

/*
 * method for saving video paramaters
 *
 * @param counterVideo the counter of Video media multimedia session
 *
 * @param divide the media param in form of var=param
 */
public void parseVideo(int counterVideo, String[] divide) {

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

        this.v.setVideoMap(divide);
        matchParameters("video", divide);
        this.session.setVideoMedia(counterVideo, v);
    }

    /*
     * method for saving audio paramaters
     *
     * @param counterVideo the counter of Audio media multimedia session
     *
     * @param divide the media param in form of var=param
     */
    public void parseAudio(int counterAudio, String[] divide) {
        this.a.setAudioMap(divide);
        matchParameters("audio", divide);
        this.session.setAudioMedia(counterAudio, a);
    }

    /*
     * method for saving data paramaters
     *
     * @param counterVideo the counter of data media multimedia session
     *
     * @param divide the media param in form of var=param
     */
    public void parseData(int counterData, String[] divide) {
        this.d.setDataMap(divide);
        matchParameters("data", divide);
        this.session.setDataMedia(counterData, d);
    }

    /*
     * Matching parsed parameters in form var=param
     *
     * @param param the pair of value and parm of media information
     *
     * @param flag -> media type of current parameters contained in param
     */
    public void matchParameters(String flag, String[] param) {
        switch (param[0]) {
            case "ip_src":
                if (flag.equals("video")) {
                    this.v.setSourceIp(param[1]);
                } else if (flag.equals("audio")) {
                    this.a.setSourceIp(param[1]);
                } else {
                    this.d.setSourceIp(param[1]);
                }
                break;
            case "ip_dst":

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

        if (flag.equals("video")) {
            this.v.setDestinationIp(param[1]);
        } else if (flag.equals("audio")) {
            this.a.setDestinationIp(param[1]);
        } else {
            this.d.setDestinationIp(param[1]);
        }
        break;
case "proto":
    if (flag.equals("video")) {
        this.v.setProtocol(param[1]);
    } else if (flag.equals("audio")) {
        this.a.setProtocol(param[1]);
    } else {
        this.d.setProtocol(param[1]);
    }
    break;
case "port_src":
    if (flag.equals("video")) {
        this.v.setSourcePort(param[1]);
    } else if (flag.equals("audio")) {
        this.a.setSourcePort(param[1]);
    } else {
        this.d.setSourcePort(param[1]);
    }
    break;
case "port_dst":
    if (flag.equals("video")) {
        this.v.setDestinationPort(param[1]);
    } else if (flag.equals("audio")) {
        this.a.setDestinationPort(param[1]);
    } else {
        this.d.setDestinationPort(param[1]);
    }
    break;
case "codec":
    if (flag.equals("video")) {
        this.v.setCodecInfo(param[1]);
    } else if (flag.equals("audio")) {
        this.a.setCodecInfo(param[1]);
    }
    break;
case "bitrate":
    if (flag.equals("video")) {
        this.v.setBitrateInfo(param[1]);
    } else if (flag.equals("audio")) {
        this.a.setBitrateInfo(param[1]);
    } else {
        this.d.setBitrateInfo(param[1]);
    }

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

        break;
    }
}

/*
 * Getter for Media Session
 */
public MediaSession getSession() {
    return this.session;
}
}

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Dodatak B - Programska izvedba funkcionalnih proširenja izvornog koda ACO zasnovanog algoritma

Sljedeći kod je dio izmjene izvornog koda ACO zasnovanog algoritma implementira procjenu gubitka paketa na poveznicama.

Razred *QoS*Routing:

```
public <Edge, Double> getPacketLossEdgeEstimations() {
    Map<Edge, Double> ret = new HashMap<Edge, Double>();

    Map<Node, Set<Edge> > edges = this.topologyManager.getNodeEdges();

    for (edge e: edges.keySet()) {
        NodeConnector i = e.getTailNodeConnector();
        NodeConnector j = e.getHeadNodeConnector();
        NodeConnectorStatistics stati =
this.statisticsManager.getNodeConnectorStatistics(i);
        NodeConnectorStatistics statj =
this.statisticsManager.getNodeConnectorStatistics(j);

        loss *= (stati.getReceivePacketCount() / statj.getTransmitPacketCount());

        ret.put(e, 1 - loss);
    }

    return ret;
}
```

```
Map<Node, Double> packetLoss = this.getPacketLossEstimations();
Map<Node, Double> delay = this.getDelayEstimations();
Map<Edge, Double> packetLossEdge = this.getPacketLossEdgeEstimations();

AntColony aco = new AntColony(edges, srcNode, dstNode,
    trafficType, packetLoss, packetLossEdge, delay);
```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Razred *AntColony*:

```

protected Map<Edge, Double> packetLossEdge;
protected double[][] nearnessEdge;

/* constructor

public AntColony(Map<Node, Set<Edge> > graph,
                Node source, Node destination, TrafficType trafficType,
                Map<Node, Double> packetLoss, Map<Node, Double> delay, Map<Edge, Double>
packetLossEdge) {

    this.packetLossEdge = packetLossEdge;

    this.nearnessEdge = new double[size][size];

    for (edge e: this.packetLossEdge.keySet()) {
        int i = node2int.get(e.getHeadNodeConnector().getNode());
        int j = node2int.get(e.getHeadNodeConnector().getNode());

        this.nearnessEdge[i][j] = this.packetLossEdge.get(e);
        this.nearnessEdge[j][i] = this.packetLossEdge.get(e);
    }

    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            this.pheromones[i][j] = 0.5;
            this.nearness[i][j] = (1 - packetLoss.get(int2node.get(i))) * (1 -
packetLoss.get(int2node.get(j))) * (1 - this.nearnessEdge[i][j]); //??????
        }
    }
}

```


Integracija upravljačkog uređaja OpenDaylight u programski upravljanu laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Sljedeći kod odnosi se na izmjene izvornog algoritma te omogućuju dinamičko dodavanje krajnjih uređaja unutar logičke topologije OpenDaylight-a.

Razred *QoS*Routing:

@Override

```
public PacketResult receiveDataPacket(RawPacket inPkt) {
    /* extract IP addresses from packet data */
    Packet formattedPak = this.dataPacketService.decodeDataPacket(inPkt);
    InetAddress source = null, destination = null;
    IPv4 ipPak = null;
    if (formattedPak instanceof Ethernet) {
        Object nextPak = formattedPak.getPayload();
        System.out.println("Recieved ethernet frame");
        if (nextPak instanceof IEEE8021Q) {
            IEEE8021Q vlanPak = (IEEE8021Q) nextPak;
            Object nextNextPak = vlanPak.getPayload();
            System.out.println("Recieved VLAN packet");
            if (nextNextPak instanceof ARP) {
                System.out.println("Recieved ARP packet");
                ARP arpPak = (ARP)nextNextPak;
                Status s;
                String IpAddr, MacAddr, VlanId;
                NodeConnector nc;
                IpAddr=byteArrayToString(arpPak.getSenderProtocolAddress(),"IPv4");
                MacAddr=byteArrayToString(arpPak.getSenderHardwareAddress(),"MAC");
                VlanId=Short.toString(vlanPak.getVid());
                nc=inPkt.getIncomingNodeConnector();

                //installFlow(nc.getNode(), nc, "ARP", arpPak.getSenderHardwareAddress(), null,
false);

                try{
```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

        if(this.hostManager.hostQuery(InetAddress.getByName(IpAddr))==null){
            s = this.hostManager.addStaticHost(IpAddr, MacAddr, nc, VlanId);
            if (s.isSuccess()) {
                System.out.println("Added host: " + IpAddr + ", " + MacAddr + ", " + VlanId);
            } else {
                System.out.println("Adding host: " + IpAddr + ", " + MacAddr + ", " + VlanId
+ " failed");
            }
        }
    }
}
catch(UnknownHostException e){
    e.printStackTrace();
}
}
else if (nextNextPak instanceof IPv4) {
    ipPak = (IPv4)nextNextPak;
    System.out.println("Recieved IPv4 datagram");

    source = NetUtils
        .getInetAddress(ipPak.getSourceAddress());
    destination = NetUtils
        .getInetAddress(ipPak.getDestinationAddress());
    System.out.println(ipPak.getProtocol());
    Object ICMPPak=ipPak.getPayload();
    if(ICMPPak instanceof ICMP){
        System.out.println("Recieved ICMP packet");
        ICMP nextPak2=(ICMP) ICMPPak;

        createNewFlow(source, destination, ipPak, TrafficType.DATA, false);

    }

    //Object TCPPak = ipPak.getPayload();

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

    ///if TCP
    //if(TCPPak instanceof TCP){
        //TCP nextPak2=(TCP) TCPPak;

        //short destPort=nextPak2.getDestinationPort();
        //if(destPort==32000){
            //System.out.println("poruka za server");
            //}
        //}
    }
}
else if (nextPak instanceof IPv4) {
    ipPak = (IPv4)nextPak;

    source = NetUtils
        .getInetAddress(ipPak.getSourceAddress());
    destination = NetUtils
        .getInetAddress(ipPak.getDestinationAddress());
}
}

if (source == null || destination == null) {
    return PacketResult.IGNORED;
}

System.out.println("Received new packet!");
System.out.println("Src: " + source.getHostAddress() +
    " Dst: " + destination.getHostAddress());

/* create new a flow between source and destination */
createNewFlow(source, destination, ipPak, TrafficType.OTHER, true);

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

/* Check the packet content */
try {
    byte[] payload = null;
    Object pak = formattedPak;

    if (pak instanceof Ethernet) {
        pak = ((Ethernet) pak).getPayload();
        if (pak instanceof IPv4) {
            pak = ((IPv4) pak).getPayload();

            if (pak instanceof TCP) {
                payload = ((TCP) pak).getRawPayload();
            }
            if (pak instanceof UDP) {
                payload = ((UDP) pak).getRawPayload();
            }
        }
    }
}

/* parse UDP/TCP packet payload */
if (payload != null) {
    String data = new String(payload, "UTF-8");

    InetAddress streamAddress = parsePacketData(data);

    if (streamAddress != null) {
        System.out.println("IP: " + streamAddress.getHostAddress());

        /* create a new flow for video stream between destination
        * and stream address */
        createNewFlow(destination, streamAddress, null,
            TrafficType.VIDEO, false);
    }
}

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

```

    }

    } catch(Exception e) {
        System.out.println(e.toString());
    }

    return PacketResult.CONSUME;
}

```

Integracija upravljačkog uređaja OpenDaylight u programski upravljaju laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

Sljedeći kod odnosi se na izmjene izvornog algoritma te unaprijed definira tokova namjenjene DHCP zahtjevima i odgovorima.

Razred *QoS*Routing:

```
void installPredefinedFlows() {
    System.out.println("Installing predefined flows for all nodes!");

    /* For each SDN switch known to the controller... */
    for (Switch swc: this.switchManager.getNetworkDevices()) {
        Node node = swc.getNode();
        Set<NodeConnector> nodeConnectors = swc.getNodeConnectors();

        /* For each node connector (OpenFlow port) on SDN switch... */
        for (NodeConnector nc: nodeConnectors) {
            String nodeConnectorId = nc.getNodeConnectorIdAsString();

            /* For each OpenFlow port with label 1 in its ID... */
            if (nodeConnectorId.contains("OF|1")) {
                //byte[] macAddr = new byte[] {(byte) 0xff, (byte) 0xff, (byte) 0xff, (byte) 0xff,
                (byte) 0xff, (byte) 0xff};
                byte[] ipAddr = new byte[] {(byte) 255, (byte) 255, (byte) 255, (byte) 255};

                try {
                    InetAddress dhcpDstIpAddr = InetAddress.getByAddress(ipAddr);
                    InetAddress multicastAddr = InetAddress.getByAddress(ipAddr2);

                    /* ...install flows for DHCP requests (dest. port 67) and responses (dest.
                    port 68) */
                    installFlow(node, nc, null, dhcpDstIpAddr, "udp", (short) 0, (short) 67,
```

Integracija upravljačkog uređaja OpenDaylight u programski upravljano laboratorijsku mrežu	Verzija: 2.0
Tehnička dokumentacija	Datum: 24.01.2016.

false);

installFlow(node, nc, null, dhcpDstIpAddr, "udp", (short) 0, (short) 68,

false);

} catch (UnknownHostException uhe) {

System.out.println("Exception: " + uhe.toString());

return;

}

}

}

}

return;

}