SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 1087

# Usmjeravanje u programski upravljanoj mreži zasnovano na parametrima kvalitete višemedijske usluge

Matija Šantl

Zagreb, srpanj 2015.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 6. ožujka 2015.

# DIPLOMSKI ZADATAK br. 1087

Pristupnik: **Matija Šantl (0036458898)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Usmjeravanje u programski upravljanoj mreži zasnovano na parametrima kvalitete višemedijske usluge**

Opis zadatka:

U programski upravljanoj komunikacijskoj mreži mrežni uređaj obavlja samo funkciju prosljeđivanja tokova podataka, dok se funkcija njihovog usmjeravanja izvodi na središnjem upravljačkom uređaju. Takva arhitektura olakšava izvedbu modela usmjeravanja koji može odrediti optimalne mrežne puteve za različite tokove podataka iste višemedijske usluge, ovisno o parametrima kvalitete usluge. Specifikacija OpenFlow definira komunikacijski protokol između upravljačkog i mrežnog uređaja, kojim se zadaju pravila prosljeđivanja podataka.
Vaš je zadatak proučiti i opisati specifikaciju OpenFlow, koncept programski upravljane mreže te njezinu izvedbu u simulatoru/emulatoru mreža IMUNES. Proučite i usporedite postojeće modele i programske izvedbe algoritama usmjeravanja u komunikacijskoj mreži. Razradite i programski izvedite model usmjeravanja u programski upravljanoj komunikacijskoj mreži koji će omogućiti odabir optimalnih puteva zasnovan na vrsti toka podataka i parametrima kvalitete višemedijskih usluga, uz zadana ograničenja. Provedite vrednovanje predloženog modela usmjeravanja na odabranim studijskim slučajevima i demonstrirajte njegovu primjenu u prosljeđivanju medijskih tokova kroz programski upravljanu mrežu emuliranu alatom IMUNES.
Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 30. lipnja 2015.

Mentor:
Ognjen Dobrijević
Doc. dr. sc. Ognjen Dobrijević

Predsjednik odbora za diplomski rad profila:
Prof. dr. sc. Siniša Srbljić

Dielovođa:
Doc. dr. sc. Tomislav Hrkać

*Zahvaljujem se mentoru, doc. dr. sc. Ognjenu Dobrijeviću, na uloženom trudu i vremenu za izradu diplomskog rada te vodstvu tokom cijelog diplomskog studija i Zavodu za telekomunikacije na pruženom prostoru za izradu diplomskog rada.*

*Ovaj diplomski rad ne bi nastao da nije bilo stalne potpore moje obitelji tokom cijelog studija, hvala Željko, Draženka i Dino.*

*Posebno hvala Mirni Družinec na podršci, strpljenju, savjetima te ljubavi i brizi.*

# SADRŽAJ

# 1. Uvod

Klasične komunikacijske mreže temeljene na prethodno programiranim uređajima čije karakteristike ovise od proizvođača do proizvođača onemogućuju provedbu ispitivanja novih algoritama usmjeravanja u komunikacijskim mrežama.

Kao rješenje tog problema, predložena je specifikacija *OpenFlow* kao jedna izvedba koncepta programski upravljanih komunikacijskih mreža (engl. *Software Defined Networks, SDN*). Osnovna ideja predloženog pristupa je upravljanje prosljeđivanjem podataka u mrežnim uređajima pomoću programske podrške smještene na logički centraliziranom upravljačkom uređaju (engl. *Controller*) koji zna globalno stanje komunikacijske mreže. Apstrakcija elemenata mreže izdvajanjem funkcije upravljanja prosljeđivanjem i definiranjem komunikacijskih sučelja prema njima provodi se radi lakšeg i bržeg razvoja mrežnih usluga poput kontrole pristupa mreži, kontrole prihvata novih tokova, poboljšanja kvalitete usluge i mnogih drugih.

Cilj ovog diplomskog rada je proučiti i opisati postojeće algoritme usmjeravanja zasnovanog na kvaliteti usluge (engl. *Quality of Service*), proučiti i opisati koncept programski upravljane komunikacijske mreže zajedno sa specifikacijom *OpenFlow*, predložiti algoritam usmjeravanja zasnovanog na kvaliteti usluge za SDN te razviti proširenje za upravljački uređaj *OpenDaylight* koje programski izvodi predloženi algoritam.

U drugom je poglavlju definiran problem usmjeravanja zasnovanog na kvaliteti usluge te je dan opis i usporedba postojećih algoritama za takvu vrstu usmjeravanja. Nakon toga, u trećem je poglavlju dan opis arhitekture programski upravljanih komunikacijskih mreža i specifikacije *OpenFlow* te je dan i pregled postojećih rješenja usmjeravanja zasnovanog na kvaliteti usluge. U četvrtom je poglavlju dan osvrt na upravljački uređaj *OpenDaylight* te je uz opis korištenih alata opisano kako dodati novo proširenje u upravljački uređaj. U petom je poglavlju opisan predloženi algoritam usmjeravanja zasnovanog na kvaliteti usluge. Šesto poglavlje opisuje programsku izvedbu algoritma usmjeravanja zasnovanog na kvaliteti usluge dok se u sedmom poglavlju nalazi evaluacija predloženog algoritma. Na kraju rada nalazi se zaključak.

# 2. Usmjeravanje zasnovano na kvaliteti usluge

Kvaliteta usluge, *QoE*, predstavlja konceptualni pomak u analiziranju izvođenja usluga sa stajališta krajnjeg korisnika, pa se tako u obzir uzimaju tehnički kao i ne-tehnički faktori koji utječu na kvalitetu usluge, ponajviše multimedijskih usluga. Izuzev mrežnih parametara, *QoE* razmatra (a) parametre vezane uz aplikaciju kao npr. vrsta sadržaja i način kodiranja, (b) parametre vezane uz sustav kao npr. svojstva krajnjeg korisnika i poslužitelja, (c) parametre vezane uz korisnika kao npr. postavke kranjenjeg korisnika, i (d) kontekstualne parametre poput npr. cijene usluge.

Kako bismo definirali problem usmjeravanja zasnovanog na kvaliteti usluge potrebno je najprije definirati oznake koje ćemo koristiti prilikom opisa problema.

Neka je $G(N, E)$ topologija komunikacijske mreže pri čemu je $N$ skup čvorova, a $E$ skup bridova, tj. veza između čvorova. Kao oznake kardinaliteta skupa upotrebljavane su oznake $|N|$ za skup $N$, odnosno $|E|$ za skup $E$. Broj mjera kojima opisujemo kvalitetu usluge (npr. kašnjenje, broj skokova, itd.) označujemo s $m$. Svaki brid je opisan $m$-dimenzionalnim vektorom težina $w(u, v) = (w_1, ..., w_m)$, pri čemu su $u$ i $v$ oznake čvorova koje povezuje brid, a $w_i$ skalarna oznaka mjere kvalitete usluge, $1 \leq i \leq m$. Duljinu puta $P$ označujemo s $l(P)$.

**Definicija 1.** Problem pronalaska puta s višestrukim ograničenjima
(engl. *Multi-Constrained Path Problem, MCP*) [15]:

Neka je zadana komunikacijska mreža $G(N, E)$, dva čvora $u, v \in N$ te brid $(u, v) \in E$ koji ima vektor težina s $m$ aditivnih mjera kvalitete usluge takve da $w_i(u, v) \geq 0, i = 1, ..., m$. Ako su zadana ograničenja $L$ u obliku vektora težina dimenzija $m$, $L_i, i = 1, ..., m$, problem pronalaska puta $P$ iz izvorišta $s$ do odredišta $d$ se definira kao:

$$w_i(P) \coloneqq \Sigma_{(u,v)\in P} w_i(u, v) \leq L_i \tag{1}$$

za svaki $i = 1, ..., m$.

Moguće je da postoji više puteva u mreži $G(N, E)$ koji zadovoljavaju svih $m$ ograničenja. Prema Definiciji 1, bilo koji od tih puteva je rješenje MCP problema.

**Definicija 2.** Problem pronalaska optimalnog puta s višestrukim ograničenjima (engl. *Multi-Constrained Optimal Path Problem, MCOP*) [15]:

Neka je zadana komunikacijska mreža $G(N, E)$, dva čvora $u, v \in N$ te brid $(u, v) \in E$ koji ima vektor težina s $m$ aditivnih mjera kvalitete usluge takvih da $w_i(u, v) \geq 0, i = 1, ..., m$. Ako su zadana ograničenja $L$ u obliku vektora težina dimenzija $m$, $L_i, i = 1, ..., m$, problem pronalaska puta $P$ iz izvorišta $s$ do odredišta $d$ definira se kao:

1. $w_i(P) \coloneqq \Sigma_{(u,v)\in P} w_i(u, v) \leq L_i$ za svaki $i = 1, ..., m$
2. $l(P) \leq l(P*), \forall P*$ pri čemu $P$ zadovoljava uvjete iz (1) a $P*$ predstavlja puteve s istom težinom.

Posljednjih desetak godina pojavilo se više različitih algoritama za usmjeravanje zasnovanih na kvaliteti usluge (engl. *QoS-based routing algorithms*).

U nastavku se nalazi pregled nekih od tih algoritama.

## 2.1. Jaffeov aproksimacijski algoritam

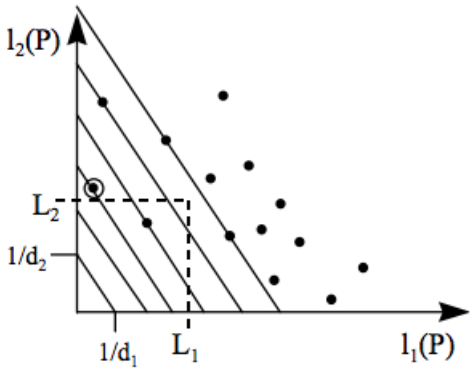
Jeffrey M. Jaffe je 1984. predstavio dva algoritma za rješavanje MCP problema [9]. Jedan od njih je egzaktan, ali zbog svoje vremenske složenosti koja je pseudo-polinomijalna nije primjenjiv u praksi, a drugi,poznatiji pod nazivom *Jaffeov algoritam*, bit će pobliže opisan.

U pojednostavljenoj verziji, Jaffe predlaže korištenje algoritama za traženje najkraćeg puta [1] pri čemu su težine između bridova linearna kombinacija mjera kvalitete:

$$w(u, v) = d_1 \cdot w_1(u, v) + d_2 \cdot w_2(u, v) \tag{2}$$

a $d_1$ i $d_2$ su pozitivni realni koeficijenti.

[1] Dijkstrin ili Bellman-Ford algoritam




**Slika 1:** Prostor pretraživanja Jaffeovog algoritma, preuzeto iz [15]

Svaka linija na Slici 1 predstavlja puteve jednake duljine s obzirom na definiciju težine kao linearnu kombinaciju mjere kvalitete. Jaffeov algoritam pretražuje prostor mogućih puteva krećući se po paralelnim linijama koje su definirane s $w(P) = c$. Pravokutnik definiran koordinatnim osima i isprekidanom crtom na oznakama $L_1$ i $L_2$, koja predstavljaju zadana ograničenja, predstavlja prostor puteva koji zadovoljavaju dana ograničenja. Čim jedna od linija naiđe na put koji je označen točkom, algoritam se zaustavlja i kao rezultat vraća najkraći put s obzirom na definiciju težine kao linearnu kombinaciju. Na Slici 1, najkraći put je prikazan kao zaokružena točka.

Slika 1 još pokazuje kako najkraći put koji se temelji na linearnoj kombinaciji ne mora biti u prostoru ograničenja. Jeffe se u definiciji algoritma na to osvrnuo davši nelinearnu definiciju duljine puta u obliku:

$$f(P) = max\left\{w_1(P), L_1\right\} + max\left\{w_2(P), L_2\right\} \tag{3}$$

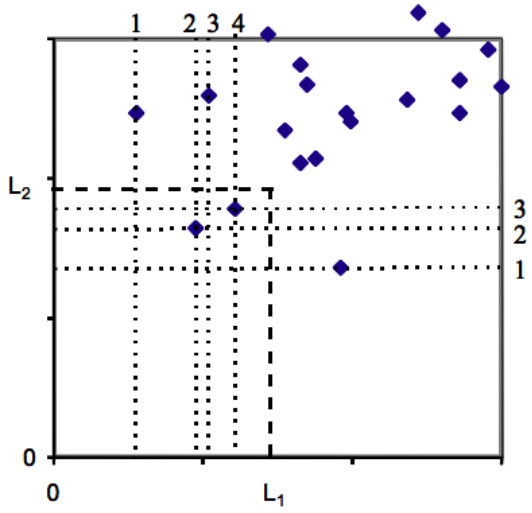
čija minimizacija garantira pronalazak puta u prostoru ograničenja, ako takav put postoji. Međutim, nedostatak nelinearne definicije duljine je što ne postoji jednostavan algoritam za pronalazak najkraćeg puta koji bi se mogao nositi s time. Upravo se zbog toga Jaffe odlučio za korištenje linearne definicije kojom aproksimira vrijednost nelinearne definicije duljine puta.

## 2.2. Iwatin algoritam

Iwata je 1996. predložio polinomijalan algoritam za rješavanje MCP problema [5]. Algoritam najprije pronalazi najkraći put s obzirom na jednu od mjera kvalitete usluge te nakon toga provjerava ako su sva ograničenja zadovoljena. Ako neko od ograničenja

nije zadovoljeno, procedura se ponavlja za drugu mjeru kvalitete usluge sve dok nisu iscrpljene sve mjere.

Nedostatak Iwatinog algoritma je što nema garancije da je neki od najkraćih puteva s obzirom na jednu mjeru kvalitete usluge blizu puta koji se nalazi unutar ograničenja.




**Slika 2:** Najkraći putevi s obzirom na mjeru kvalitete usluge, preuzeto iz [15]

Slika 2 prikazuje nedostatak Iwatinog algoritma. Na osi apscisa su prikazane duljine puteva s obzirom na mjeru $m_1$, a na osi ordinata duljine puteva s obzirom na mjeru $m_2$. Pravokutnik ograđen isprekidanom linijom označava prostor puteva koji zadovoljavaju dana ograničenja, $L_1$ i $L_2$. Tek se drugi i četvrti put za mjeru $m_1$ te drugi i treći za mjeru $m_2$ nalaze u prostoru ograničenja, što znači da iako postoje rješenja, Iwatin algoritam ne bi pronašao niti jedno.

## 2.3. TAMCRA

Hans de Neve i Piet Van Mieghem su 2000. predstavili algoritam TAMCRA *Tunable Accuracy Multiple Constraint Routing Algorithm* [13]. TAMCRA se temelji na tri osnovna načela: nelinearna mjera za duljinu puta, pristup k-najkraćih puteva i princip ne-dominirajućih puteva [15].

Svih $m$ mjera kvalitete usluge se smatra jednako značajnima te moraju biti aditivne. Svaki brid, odnosno veza dva čvora, je opisan s $m$-dimenzionalnim vektorom težina $(w_1, ..., w_m)$. Vektor puta $w_j(P) = \{w_1(P), w_2(P), ..., w_m(P)\}$ je zbroj vektora težina bridova koji se nalaze na putu $P$, $w_j(P) = \Sigma_{i \to l \in P} w_j(i \to l)$ za svaki $j$. Duljina puta

$P$ je norma vektora zadana s:

$$l(P) = max_{1 \leq j \leq m}(w_j(P)/L_j) \tag{4}$$

Ideja iza ne-linearne reprezentacije duljine puta opisana je u poglavlju 2.1.

Pristup k-najkraćih puteva je opravdan sljedećim primjedbama. Kod traženja najkraćeg puta u više dimenzija princip optimalnosti nije ostvaren, tj. svaka podsekvenca optimalne sekvence odluka nije optimalno rješenje pripadajućeg potproblema, te zbog toga algoritmi za traženje najkraćeg puta s jednim ograničenjem (Dijkstra, Bellman-Ford, Floyd-Warshall i drugi) neće uvijek davati točne rezultate. Prilikom izvršavanja tih algoritama, oni pretpostavljaju princip optimalnosti te spremaju međurezultat do svakog čvora na putu. Budući da postoje višestruka ograničenja te se koristi nelinearna mjera za duljinu puta, najkraći put do nekog međučvora neće nužno biti dio najkraćeg puta do odredišnog čvora. Upravo se zbog toga za svaki čvor sprema više od jednog puta, te time dolazimo na pristup k-najkraćih puteva [13].

Cjelobrojni parametar $k$ se prirodno nameće kao indeks podešavanja točnosti. Što je veća vrijednost parametra $k$ to je veća vjerojatnost da će najkraći put od izvorišta do međučvora biti dio najkraćeg puta od izvorišta do odredišta.

Treća ideja iza algoritma TAMCRA koja značajno povećeva efikasnost je korištenje *dominirajućih puteva*, koncept kojeg je prvi predstavio Henig 1985. godine [13]. "Dominacija puta" se može promatrati kao višedimenzionalna relaksacija [2].

**Definicija 3.** Put $P$ se zove ne-dominirajući put ako ne postoji put $P'$ za koji vrijedi $w_i(P) \leq w_i(P')$ za sve komponente vektora težina $i$, osim za barem jednu komponentu $j$ za koju vrijedi $w_j(P') < w_j(P)$.

Neka su dana dva puta, $P_1$ i $P_2$ iz izvorišta do nekog međučvora, te neka promatramo dvije mjere kvalitete usluge, $m = 2$. Označimo duljine puteva $P_1$ i $P_2$ kao $w_1(P_1), w_2(P_1) = (x_1, y_1)$ odnosno $w_1(P_2), w_2(P_2) = (x_2, y_2)$. Promatrajmo dva slučaja.

1. $P_1$ je kraći od $P_2$ i $w_i(P_1) < w_i(P_2), \forall i$. U tom slučaju svaki put od izvorišta do odredišta koji sadrži $P_1$ će biti kraći od bilo kojeg drugog puta od izvorišta do odredišta koji sadrži $P_2$.

2. Neka vrijedi $w_i(P_1) < w_i(P_2)$ za neke indekse $i$, ali također vrijedi $w_j(P_1) > w_j(P_2)$ za barem jedan indeks $j$. U tom slučaju najkraći put od izvorišta do nekog međučvora nije nužno sadržan u najkraćem putu od izvorišta do odredišta.

[2] Poput relaksacije koja se radi kod algoritama Dijkstre i Bellman-Ford

Ako se prilikom obilaska nekog čvora dogodi slučaj (2), onda se svih $m$ komponenti za oba puta sprema u red čvora.

## 2.4. SAMCRA

Hans de Neve, Piet Van Mieghem i Fernando Kuipers su 2001. predstavili poboljšanje algoritma TAMCRA pod nazivom SAMCRA *Self Adaptive Multiple Constraints Routing Algorithm* [14].

Dok je kod algoritma TAMCRA broj puteva koji se uzimaju u obzir prilikom obilaska međučvorova fiksiran i iznosi $k$, algoritam SAMCRA sam prilagođava parametar $k$, koji u najgorem slučaju može rasti i eksponencijalno. Gornja granica koju parametar $k$ može postići iznosi $k_{max} = \lfloor e(N-2)! \rfloor$, što je zapravo ukupan broj puteva između izvorišta i odredišta u mreži $G(N, E)$ [15].

Prilagodba parametra $k$ čini algoritam SAMCRA egzaktnim rješenjem MCOP problema. Algoritam SAMCRA garantira pronalazak najkraćeg puta unutar ograničenja ako takav put postoji.

## 2.5. Chenov aproksimacijski algoritam

Chen i Nahrstedt su 1998. godine preložili aproksimativan algoritam za rješavanje MCP problema [15]. Algoritam najprije transformira MCP problem u jednostavniji problem tako da skalira $m-1$ realnih vrijednosti mjera kvalitete usluge u cjelobrojne vrijednosti na sljedeći način:

$$w_i^*(u,v) = \left\lceil \frac{w_i(u,v) \cdot x_i}{L_i} \right\rceil, \forall i \in [2, ..., m] \tag{5}$$

pri čemu su $x_i$ predefinirani pozitivni cijeli brojevi. Pojednostavljeni problem se sastoji od traženja puta $P$ za koji $w_i(P) \leq L_i$ i $w_i^*(P) \leq x_i, \forall i \in [2, ..., m]$. Rješenje tog problema je također i rješenje originalnog MCP problema, ali obrat ne vrijedi. Budući da je pojednostavljeni problem rješiv egzaktno, Chen i Nahrstedt su pokazali da se MCP problem može riješiti u polinomijalnom vremenu pod uvjetom da barem $m-1$ mjera kvalitete usluge imaju ograničene cjelobrojne vrijednosti.

Za rješavanje pojednostavljenog MCP problema predložili su dva algoritma koji se temelje na dinamičkom programiranju:

– Prošireni Dijkstrin algoritam za pronalazak najkraćeg puta (engl. *EDSP - Extended Dijkstra's Shortest Path algorithm*)

– Prošireni Bellman-Ford algoritam (engl. *EBF - Extended Bellman-Ford algorithm*).

Algoritmi kao rezultat vraćaju put koji minimizira realnu (prvu) mjeru kvalitete usluge, dok je ostalih $m - 1$ cjelobrojnih mjera unutar ograničenja.

## 2.6. Pseudo-slučajan algoritam

Korkmaz i Krunz su 2001. predložili pseudo-slučajan heuristički algoritam za rješavanje MCP problema [12]. Ideja koja stoji iza pseudo-slučajnosti je da se odluke tijekom izvršavanja algoritma donose slučajno kako bi se izbjegle potencijalne nepredvidive zapreke prilikom pretraživanja.

Algoritam je podijeljen u dva dijela: inicijalizacijski dio i pseudo-slučajno pretraživanje. U inicijalizacijskom dijelu, algoritam računa najkraći put za svaki čvor $u$ do odredišnog čvora $d$ s obzirom na svaku mjeru kvalitete usluge i s obzirom na linearnu kombinaciju svih $m$ mjera. Ta će informacija biti korištena kao donja granica za težinski vektor puta svih puteva od $u$ do $d$. Temeljem te informacije, algoritam može odlučiti postoji li šansa za pronalazak puta s obzirom na dana ograničenja. Ako postoji, algoritam započinje s čvorom izvorišta i obilazi danu topologiju koristeći pseudo-slučajno pretraživanje u širinu (engl. *randomized breadth first search*). Koristeći informacije iz faze inicijalizacije, pseudo-slučajno pretraživanje u širinu provjerava postoji li šansa da se dosegne odredište $d$ iz nekog čvora prije samog obilaska tog čvora. Na taj način se predviđaju potencijalne zapreke i izbjegava obilazak takvih čvorova. Takva tehnika smanjivanja prostora pretraživanja se naziva svojstvo gledanja unaprijed (engl. *look-ahead property*). Pseudo-slučajno pretraživanje u širinu nastavlja pretragu tako da slučajno odabire otkrivene čvorove dok se ne dosegne odredište. Ako pretraga ne rezultira pronađenim rješenjem, dovoljno je ponovno pokrenuti pseduo-slučajno pretraživanje u širinu kako bi se povećala vjerojatnost pronalaska puta.

## 2.7. H_MCOP

Korkmaz i Krunz su 2001. također predstavili i heuristiku koju su nazvali H_MCOP [11]. Ta heuristika pokušava pronaći put unutar ograničenja korištenjem nelinearne definicije duljine puta koja se koristi u algoritmu SAMCRA. Dodatno, H_MCOP istovremeno pokušava minimizirati težinu jedne jedine mjere kvalitete usluge na putu. Kako bi se istovremeno ostvarila oba cilja, H_MCOP izvršava izmijenjenu verziju

Dijkstrinog algoritma dvaput, jednom prema naprijed, od izvorišta prema odredištu, a drugi puta prema natrag, od odredišta prema izvorištu.

U smjeru prema natrag, H_MCOP koriti Dijkstrin algoritam kako bi pronašao najkraći put za svaki čvor prema odredišnom čvoru $d$ s obzirom na $w(u,v) = \Sigma_{i=1}^{m} \frac{w_i(u,v)}{L_i}$. Nadalje, te se duljine puteva koriste kako bi se procijenila prikladnost puta od nekog čvora do odredišnog čvora.

U smjeru prema naprijed, H_MCOP koristi promijenjenu verziju Dijkstrinog algoritma koja pretragu započinje u izvorišnom čvoru $s$ i obilazi čvor $u$ na temelju puta $P$, pri čemu je $P$ heuristički određen put od izvorišta $s$ do odredišta $d$ koji se dobije spajanjem trenutnog puta od izvorišta $s$ do čvora $u$ i procjene ostatka puta od čvora $u$ do odredišta $d$.

Budući da H_MCOP u obzir uzima cijele puteve od izvorišta $s$ do odredišta $d$ prije dolaska na odredište $d$, može predvidjeti ako postoje putevi koji se nalaze izvan ograničenja tijekom pretraživanja. Iako je to pravilo veoma slično svojstvu gledanja unaprijed, ova tehnika pruža samo pravilo prednosti odabira puta i ne može se koristiti kao tehnika smanjivanja prostora pretraživanja.

## 2.8. Heuristika ograničenog puta

Yuan je 2002. godine predložio dvije heuristike za rješavanje MCP problema [18]. Heuristiku ograničene granularnosti te heuristiku ograničenog puta (engl. *LPH - Limited Path Heuristics*).

Heuristika ograničene granularnosti je veoma slična Chenovom aproksimacijskom algoritmu pa se neće dalje razmatrati. Heuristika ograničenog puta je proširenje algoritma Bellman-Ford koja koristi dva osnovna koncepta algoritma TAMCRA, koncept k-prvih puteva te princip ne-dominirajućih puteva. Za razliku od algoritma TAMCRA, koji koristi k-najkraćih puteva koje sprema za svaki čvor, heuristika ograničenog puta sprema prvih *k* puteva za svaki čvor. Nadalje, heuristika ograničenog puta ne provjerava da li se trenutno razmatran put nalazi unutar ograničenja, već to provjerava samo kada se zna cijeli put od izvorišta do odredišta.

## 2.9. A* podrezivanje

Liu i Ramakrishnan su 2001. promatrali problem pronalaska više $(K)$ najkraćih puteva koji se nalaza unutar ograničenja. Funkcija duljine puta je ista kao kod Jaffeovog algo-

ritma. Autori su predložili egzaktan algoritam pod nazivom A* podrezivanje (engl. *A* Prune*) [7]. Ako ne postoji $K$ puteva koji se nalaze unutar ograničenja, algoritam kao rezultat vraća samo onih $Q$ koji se nalaze unutar ograničenja.

Za svaku mjeru kvalitete usluge, A* podrezivanje najprije traži najkraći put od izvorišta $s$ prema svim čvorovima $i \in N \setminus \{s\}$ i od odredišta $d$ prema svim čvorovima $i \in N \setminus \{d\}$. Težine tih puteva će se koristiti za evaluaciju određenog podskupa puta, tj. li je moguće nastaviti trenutni put do odredišnog čvora $d$ a da se i dalje put nalazi unutar ograničenja. Nakon faze inicijalizacije, algoritam nastavlja obilazak čvorova poput Dijkstrinog algoritma. Čvor koji ima najmanju predviđenu duljinu puta se uzima iz hrpe te se razmatraju svi njegovi susjedi. Susjedi koji uzrokuju petlju ili vode prema narušavanju ograničenja se podrezuju.

## 2.10. Usporedba opisanih algoritama

Na temelju rezultata simulacija iz [15], u nastavku je dan sažetak prednosti opisanih algoritama za usmjeravanje zasnovano na kvaliteti usluge.

1. *Obilazak čvorova poput Dijkstrinog algoritma s nelinearnom definicijom duljine puta*

   Nelinearna definicija duljine puta zahtjev je za točnost rezultata. Kada su mjere kvalitete usluge pozitivno korelirane, linearna definicija puta mogla bi imati bolju stopu pronalaska rješenja, ali pod drugim uvjetima pronađeni put može narušavati sva ograničenja.

   Implementacija inspirirana Dijkstrinim algoritmom obično je puno brža od algoritama koji obilaze čvorove poput Bellman-Fordovog algoritma.

2. *Podesiva točnost koristeći pristup k-najkraćih puteva*

   Usmjeravanje s višestrukim ograničenjima može zahtijevati da se više puteva sprema za svaki čvor, što rezultira pristupom k-najkraćih puteva.

3. *Smanjivanje prostora pretraživanja koristeći princip ne-dominirajućih puteva*

   Smanjivanje prostora pretraživanja uvijek je poželjno jer smanjuje potrebno vrijeme za izvršavanje algoritma.

4. *Predviđanje ostvarivosti puta (svojstvo gledanja unaprijed)*

   Najprije računajući put u polinomijalnom vremenu između izvorišta i odredišta, te kasnije koristeći tu informaciju za traženje ostvarivih puteva između izvorišta i

odredišta veoma je korisno, pogotovo kada topologije postanu "teško rješive", tj. $N$, $E$ i $m$ su veliki. Svojstvo gledanja unaprijed omogućava postavljanje donjih granica prilikom pretrage, što onda omogućava predviđanje ostvarivosti puta.

U nastavku je dana tablica s pregledom algoritama i pripadne vremenske složenosti.

**Tablica 1:** Vremenske složenosti algoritama za usmjeravanje zasnovano na kvaliteti usluge

| **Algoritam** | **Složenost** |
|---|---|
| Jaffeov aproksimativni algoritam | $O(NlogN + mE)$ |
| Iwatin algoritam | $O(mNlogN + mE)$ |
| TAMCRA | $O(kNlog(kN) + k^2mE)$ |
| SAMCRA | $O(kNlog(kN) + k^2mE)$ |
| Chenov aproksimacijski algoritam | $O(x_2^2 \cdots x_m^2 N^2)$ i $O(x_2 \cdots x_m NE)$ |
| Pseudo-slučajan algoritam | $O(mNlogN + mE)$ |
| H_MCOP | $O(NlogN + mE)$ |
| Heuristika ograničenog puta | $O(k^2NE)$ |
| A* podrezivanje | $O(QN \cdot (m + h + logQ))$ |

Vremenske složenosti navedenih algoritama trebaju se uzimati s oprezom. Tako će, prema simulacijama iz [15], izvođenje *H_MCOP* algoritma uvijek biti duže od *Jaffeovog* algoritma. Pokazalo se da algoritmi zasnovani na *TAMCRA* algoritmu koji koriste pristup *k* najkraćih puteva, nelinearnu funkciju težine, eliminaciju dominirajućih puteva i druge metode smanjivanja prostora pretraživanja poput gledanja-unaprijed, imaju najbolje performanse. Vremenska složenost i performanse takvih algoritama mogu se prilagođavati parametrom *k*. Kada parametar *k* nije ograničen, ti algoritmi vode k egzaktnom rješenju.

Motiviran potencijalom programski upravljanih komunikacijskih mreža te inspiriran gore navedenim algoritmima, u nastavku ovog rada opisao sam vlastiti pristup problemu pronalaska puta s višestrukim ograničenjima heurističkim algoritmom.

# 3. Programski upravljane komunikacijske mreže

Programski upravljane komunikacijske mreže (engl. *Software Defined Networks, SDN*) predstavljaju arhitekturu komunikacijskih mreža u kojoj je upravljački sloj mreže izdvojen iz pojedinačnih mrežnih uređaja i smješten u središnji upravljački uređaj. Osnovna ideja programski upravljanih komunikacijskih mreža je razdvajanje upravljanja mrežom od prosljeđivanja podataka koji prolaze kroz mrežu [6].

Klasične komunikacijske mreže temeljene na prethodno programiranim uređajima čije karakteristike ovise od proizvođača do proizvođača onemogućuju provedbu ispitivanja novih komunikacijskih protokola u produkcijskim mrežama.

Kao rješenje tog problema predložen je koncept SDN te specifikacija *OpenFlow* kao jedna od izvedbi programski upravljanih komunikacijskih mreža. Odvajanje upravljanja mrežom i usmjeravanja omogućuje komunikacijskoj mreži da postane izravno programirljiva, a pripadna se infrastruktura može prikazati apstraktno za aplikacije i mrežne usluge. Programski upravljane komunikacijske mreže time čine dinamičnu, upravljivu, prilagodljivu i isplativu arhitekturu koja pogoduje promjenjivim zahtjevima mrežnih usluga.

Svojstva arhitekture programski upravljanih komunikacijskih mreža su:

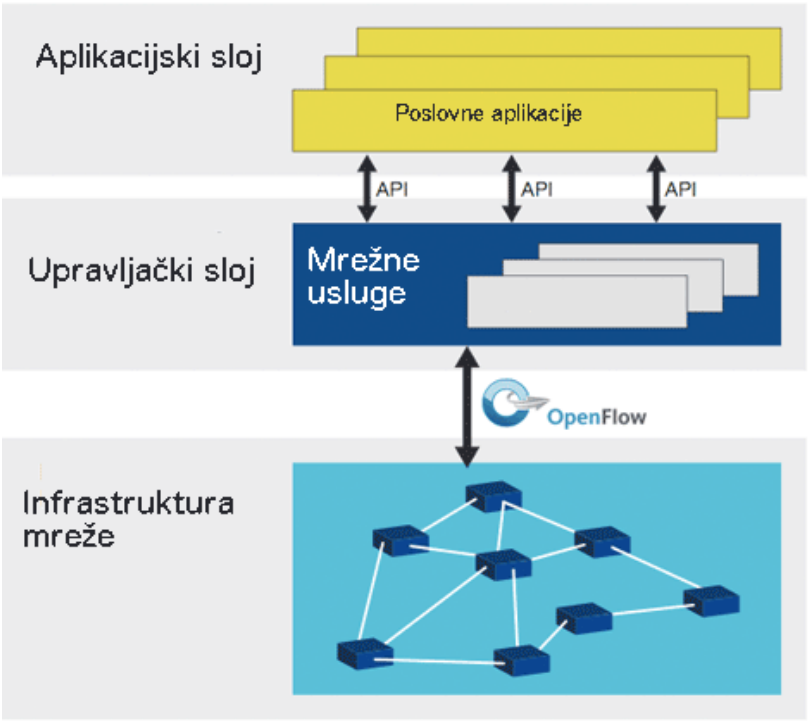
- **Izravna programirljivost**: upravljanje komunikacijskom mrežom izravno je programirljivo putem sučelja koja izlažu upravljački uređaji.
- **Agilnost**: apstrakcija odvojenosti upravljanja i usmjeravanja dozvoljava administratorima dinamične prilagodbe promjene toka podataka u komunikacijskoj mreži.
- **Centraliziranost**: logika upravljanja mrežom je centralizirana pomoću programirljivih upravljačkih uređaja koji znaju globalno stanje mreže.
- **Konfigurabilnost**: moguće je konfigurirati, upravljati, zaštiti i optimirati mrežne resurse pomoću programa upravljačkih uređaja.

– **Neovisnost o proizvođaču**: otvoreni standardi pojednostavljuju dizajn mreža i operacija zbog instrukcija koje pružaju upravljački uređaji a koji nisu specifični za proizvođača uređaja ili protokola.

Programski definirane komunikacijske mreže odlikuju se primjerice brzim oporavkom pri ispadu neke mrežne poveznice ili mrežnog uređaja jer u vrlo kratkom vremenskom roku upravljački uređaj može odrediti nova pravila prosljeđivanja u mreži [16].

## 3.1. Arhitektura

Prethodno navedena svojstva programski upravljanih komunikacijskih mreža ostvarena su različitim komponentama arhitekture [6], kao što je to prikazano na slici 3.




**Slika 3:** Arhitektura programski upravljanih komunikacijskih mreža

Aplikacijski sloj obuhvaća poslovne i mrežne aplikacije koje koriste mrežne usluge. Aplikacije komuniciraju s upravljačkim slojem preko vlastitih razvojnih sučelja, poput, npr., *Representational State Transfer, REST* sučelja. Upravljački sloj se sastoji od mrežnih usluga koje se pokreću i rade na zasebnom upravljačkom uređaju koji ima nadzor nad cijelom mrežom. Upravljački sloj vrši prilagodbu mreže, manipulaciju nad putevima usmjeravanja i prosljeđivanja paketa, i dodjeljuje kontrolu nad tokovima po-

dataka u sloju infrastrukture u skladu sa zahtjevima aplikacijskog sloja. Upravljački uređaj komunicira s mrežnim uređajima iz sloja infrastrukture preko upravljačkog sučelja definiranog npr. sa specifikacijom *OpenFlow*, koja je detaljnije opisana u poglavlju 3.2.

Sloj infrastrukture sastoji se od mrežnih uređaja koji su apstrahirani prema gornjim slojevima arhitekture kako bi se pojednostavila funkcionalnost i upravljanje mrežom.

Objasnimo dva sučelja čiji se nazivi često pojavljuju kod opisa arhitektura računala, pa tako i komunikacijskih mreža. To su *northbound* i *southbound* sučelja.

*Northbound* sučelje je sučelje koje određenoj komponenti programski upravljane komunikacijske mreže omogućuje komunikaciju s komponentama višeg sloja. Ono opisuje prostor protokolom podržane komunikacije između upravljačkog uređaja i aplikacije ili upravljačkih programa višeg sloja.

*Southbound* sučelje je sučelje koje određnoj komponenti programski upravljane komunikacijske mreže omogućuje komunikaciju s komponentama nižeg sloja. Tako je, npr., specifikacija *OpenFlow* zapravo *southbound* sučelje programski upravljanih komunikacijskih mreža, dok su *northbound* sučelja specifična verzijama upravljačkih uređaja jer su ostvarena uvođenjem aplikacijskih programskih sučelja.

## 3.2. Specifikacija OpenFlow

Najpoznatiji protokol kojeg koriste programski upravljane komunikacijske mreže za komunikaciju s mrežnim uređajima definiran je specifikacijom *OpenFlow* [6]. Osim protokola, specifikacija *OpenFlow* definira i komponente i osnovni skup funkcionalnosti koje svaka programski upravljana komunikacijska mreža mora podržavati.

*OpenFlow* pruža otvoreni protokol za upravljanje tablicama tokova (engl. *Flow Table*) u mrežnim uređajima. U nekim od scenarija korištenja mrežni administrator može odvojiti promet koji se koristi u produkciji i promet koji se koristi za istraživanje. Na taj se način dodatno pojednostavljuje ispitivanje novih algoritama za usmjeravanje, sigurnosnih modela i slično.

Svaki mrežni uređaj koji podržava specifikaciju *OpenFlow* sadrži tablicu tokova. Ti uređaji iskorištavaju činjenicu da većina suvremenih mrežnih uređaja koristi tablicu tokova za ostvarenje vatrozida (engl. *firewall*), *Network Address Translation, NAT*-a, kvalitete usluge (engl. *Quality of Service*) i prikupljanje statistika. Iako je tablica tokova različita od proizviđača do proizvođača, *OpenFlow* koristi zajednički skup funkcionalnosti koji je prisutan kod većine mrežnih uređaja.

Tri osnovna dijela *OpenFlow* mrežnog uređaja su:

1. **Tablica tokova**:

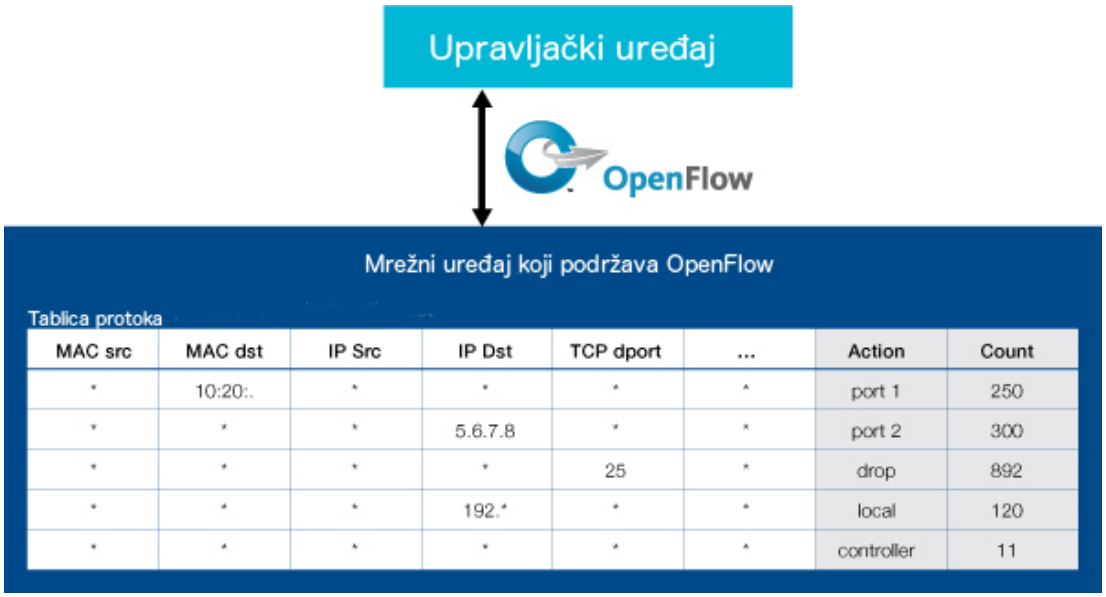
   Svakom unosu u tablici pridružena je akcija koja opisuje radnju koju je potrebno izvršiti nad pojedinim paketom. Ako se za neki dolazni paket ne pronađe odgovarajuća radnja, mrežni uređaj šalje upit upravljačkom uređaju kako bi saznao što treba učiniti s tim paketom.

2. **Siguran komunikacijski kanal**:

   Koristi se za povezivanje mrežnih uređaja s upravljačkim uređajem. Riječ je o *Secure Socket Layer/Transport Layer Security, SSL/TLS* vezi koja pomoću transportnog protokola prenosi poruke protokola definiranog specifikacijom *OpenFlow* između upravljačkog uređaja i mrežnih uređaja.

3. **Podrška za *OpenFlow* protokol**:

   Definira komunikaciju između mrežnog i upravljačkog uređaja.




**Slika 4:** Primjer tablice toka

## 3.3. Pregled postojećih rješenja usmjeravanja zasnovanog na kvaliteti usluge

U trenutnom "Internetu", dinamički mijenjati mrežna usmjeravanja na razini toka nije lagan zadatak. Prilikom dolaska paketa na mrežni uređaj, on uspoređuje par IP adresa izvorišta i odredišta paketa s unosima u tablici usmjeravanja, te na temelju, obično

preddefiniranih, pravila donosi odluku o prosljeđivanju. *OpenFlow* nudi novu paradigmu koja taj nedostatak rješava na način da dopušta mrežnom administratoru dinamičko definiranje različitih tipova tokova te njihovo povezivanje sa skupom pravila prosljeđivanja podataka. Upravljački uređaj je najvažniji element koji može donositi odluke o usmjeravanju na razini toka te osvježavati zapise u tablicama tokova.

Kako bi se osigurala optimalna kvaliteta usluge za višemedijsku uslugu s kraja na kraj mreže, prikupljanje globalnih podataka o stanju mreže poput kašnjenja i gubitka paketa je od velike važnosti [3].

Dodatno, upravljački uređaji koji podržavaju usmjeravanje zasnovano na kvaliteti usluge trebali bi imati sljedeće funkcionalnosti:

1. *Upravljanje tokovima* (engl. *Flow Management*)

   Ova funkcionalnost zadužena je za upravljanje definicijama tokova nastalih usmjeravanjem zasnovanim na kvaliteti usluge.

2. *Upravljanje redovima* (engl. *Queue Management*)

   Ova funkcionalnost pruža podršku usmjeravanju zasnovanom na kvaliteti usluge pomoću prioritetizacije redova.

3. *Upravljanje zahtjevima* (engl. *Call Admission*)

   Ova funkcionalnost odbija zahtjeve koji traže kvalitetu usluge koju je nemoguće ostvariti u komunikacijskoj mreži.

4. *Provjeravanje prometa* (engl. *Traffic Policing*)

   Ova funkcionalnost je zadužena za provjeru ostvarene kvalitete usluge podatkovnih tokova sa zatraženim parametrima kvalitete usluge.

U proteklih nekoliko godina pojavio se veliki broj istraživačkih radova na temu programski upravljanjaih komunikacijskih mreža koji se bave algoritmima i platformama za efikasno razvijanje usluga za usmjeravanje, ali još uvijek postoji potreba za algoritmom koji će u obzir uzimati različite parametre mreže kako bi maksimizirao kvalitetu usluge prema krajnjem korisniku.

Radni okvir za usmjeravanje video prometa s obzirom na kvalitetu usluga prikazan je u [4]. Model usmjeravanja prikazan je kao problem ograničenog najkraćeg puta, pri čemu je metrika strukturirana kao težinski zbroj gubitka paketa i varijance kašnjenja paketa.

Proširenje prethodnog rada, raspodijeljena upravljačka ravnina za programski upravljane komunikacijske mreže temeljena na kvaliteti usluga preko više domena, opisana

je u [3]. Tamo je model usmjeravanja temeljen na kvaliteti usluge riješen koristeći Lagrangeovu metodu opuštanja ukupnog troška.

Agarwal i drugi, u [1], predlažu rješavanje problema optimalnog usmjeravanja upotrebom polinomijalne metode aproksimacije za traženje puteva koja minimizira najveći stupanj iskorištenosti mrežnih veza.

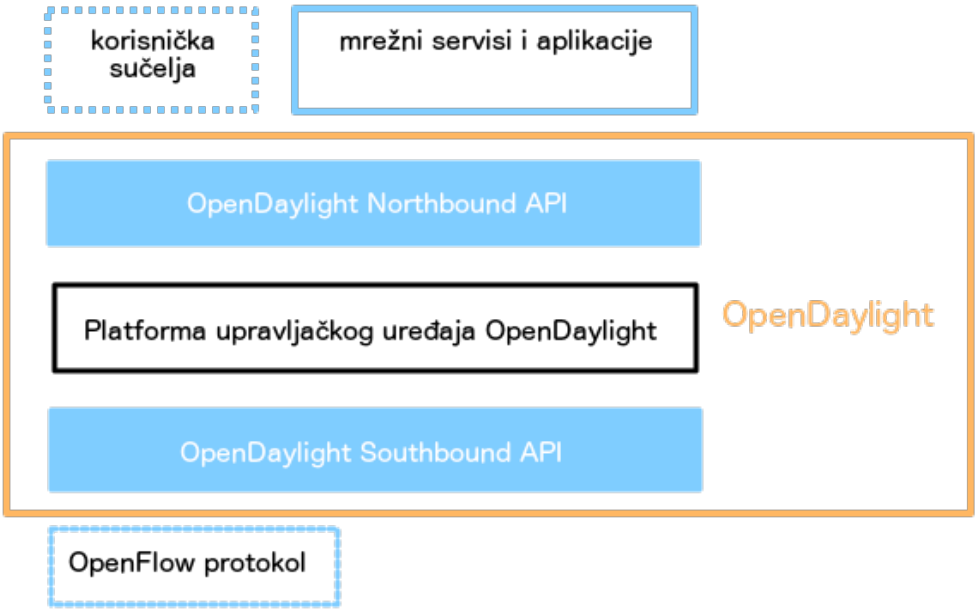
Radni okvir pravednosti kvalitete usluge (engl. *QoS Fairness Framework*) za prilagodljiv prijenos video prometa (engl. *Adaptive Video Streaming, AVS*) predstavljen je u [8]. On pravedno maksimizira količinu mrežnih resursa za postojeće klijente koristeći model kvalitete koji preslikava brzinu prometa na vrijednosti kvalitete usluge, te koristeći algoritam za raspodjelu propusnosti pronalazi optimalan skup brzina prometa za sve klijente.

Jarschel i drugi predlažu odabir puta u pristupnim mrežama koji se temelji na promatranju videa na razini međuspremnika i manifestacijama odugovlačenja za internet uslugu *YouTube* [10].

Prema svim spoznajama, algoritam za usmjeravanje temeljeno na kvaliteti usluge koji je opisan u ovom radu, razlikuje se od gore navedenih rješenja na način da u obzir uzima druge modele kvalitete usluge, koji se odnose na raznolike usluge i tipove tokova, i koristi mjeru kvalitete usluge prilikom traženja puteva.

# 4. Upravljački uređaj OpenDaylight i programska proširenja

Upravljački uređaj u programski upravljanoj komunikacijskoj mreži čini "mozak" te mreže [6]. Upravljački uređaji su strateške točke komunikacijske mreže te oni prenose informacije mrežnim uređajima koristeći *southbound* API, ali i aplikacijama koristeći *northbound* API.




**Slika 5:** Arhitektura upravljačkog uređaja OpenDaylight

*OpenDaylight* je upravljački uređaj otvorenog koda koji je napisan u programskom jeziku *Java*, te zbog toga može pokretati na svim uređajima i operacijskim sustavima koji podržavaju *Javu*. Platforma upravljačkog uređaja *OpenDaylight* (5) sadrži određene jezgrene pakete, gdje svaki od njih obavlja jednu od ključnih zadaća. Svaki se jezgreni paket može dinamički uključiti i isključiti kako bi izvršavao određene mrežne

zadaće. Postoji niz takvih osnovnih mrežnih zadaća unutar upravljačkog uređaja *OpenDaylight* kao, npr., otkrivanje topologije mreže i mogućnosti uređaja koji se nalaze u toj mreži, spremanje statistika o mrežnim uređajima te pružanje sučelja za pristup tim statistikama, i mnogi drugi.

Usluge orijentirane prema platformi te druga proširenja također se mogu ugraditi u upravljački uređaj radi poboljšanja ili vršenja eksperimenata programski definiranih komunikacijskih mreža.

## 4.1. *OSGi* specifikacija

*OSGi* (*Open Services Gateway initiative*) specifikacija opisuje modularan sustav i platformu usluga za programski jezik *Java*, koja implementira dinamičan model komponenti proširenja nekog sustava. Aplikacije ili komponente, koje dolaze u obliku proširenja, mogu se pokrenuti, zaustaviti, ažurirati te ukloniti bez potrebe ponovnog pokretanja cijelog sustava.

*OSGi* specifikacija se konceptualno može podijeliti na sljedeća područja:

- **Paket** (engl. *Bundle*)

  Uobičajeno su to *jar* komponente koje implementiraju programska proširenja s dodatnim zaglavljom manifesta.

- **Usluge** (engl. *Services*)

  Sloj usluga povezuje pakete na dinamičan način tako da pruža model objavi-pronađi-poveži za POIJ (*Plain Old Java Interfaces*) ili POJO (*Plain Old Java Objects*).

- **Registar usluga** (engl. *Services Registry*)

  API za upravljanje uslugama odnosno proširenjima koje nude paketi.

- **Životni ciklus** (engl. *Life cycle*)

  API za upravljanje životnim ciklusima (uvođenje, pokretanje, zaustavljanje, ažuriranje) paketa.

- **Moduli**

  Sloj koji definira enkapsulaciju i deklaraciju ovisnosti paketa.

- **Sigurnost** (engl. *Security*)

  Sloji koji skrbi o aspektima sigurnosti tako da ograničava funkcionalnost komponenti na prethodno definirane mogućnosti.

Upravljački uređaj *OpenDaylight* primjer je sustava koji je izgrađen na temelju *OSGi* specifikacije kako bi omogućio jednostavno dodavanje jezgrenih komponenti i dodatnih uređaja, te time, bez potrebe ponovnog pokretanja cijelog sustava, lako promijeniti način rada upravljačkog uređaja. Prilikom pokretanja upravljačkog uređaja *OpenDaylight*, nakon inicijalizacije jezgrenih dijelova, dobijamo pristup *OSGi* konzoli pomoću koje možemo dodavati nove pakete, utjecati na njihov životni ciklus i sl.

Razvoj *OSGi* komponenti za upravljački uređaj *OpenDaylight* djeluje kao dosta težak zadatak. *OpenDaylight* koristi višestruke razvojne alate i tehnike poput *OSGi* specifikacije te alata *Maven* i *git*. Sama struktura projekta dosta je kompleksna, a broj *Java* razreda je velik.

## 4.2. Slojevita apstrakcija usluge

Unutar upravljačkog uređaja *OpenDaylight* postoje dvije vrste slojevite apstrakcije usluga (engl. *Service Abstraction Layer, SAL*), 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*). *AD-SAL* koristi aplikacijsko programsko sučelje koje je statički definirano prilikom prevođenja izvornog koda usluge, a proizvođač i potrošač su ugrađeni izravno u izvorni kod. *MD-SAL* pak generira aplikacijsko programsko sučelje iz modela prilikom prevođenja izvornog koda usluge te učitava isto u upravljački uređaj prilikom učitavanja *OSGi* proširenja. Svi pozivi i podaci između proizvođača i potrošača prolaze kroz centralnu bazu podataka.

U nastavku teksta podrazumijevamo korištenje *AD-SAL*.

## 4.3. Alat *git*

*Git* je sustav za raspodijeljeno upravljanje verzijama. Namijenjen je za veće projekte koji imaju više autora, koji rade na istim dijelovima projekta. U nastavku je dan programski isječak za dohvaćanje izvornog koda upravljačkog uređaja *OpenDaylight* (programski odsječak 1).

**Programski odsječak 1** Dohvaćanje *git* repozitorija za *OpenDaylight*

```
1: $ git clone https://github.com/opendaylight/controller
```

## 4.4. Alat *Maven*

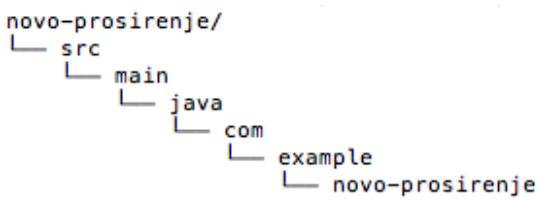
Alat *Maven* služi za upravljanje i razumijevanje programskih projekata. *Maven* upravlja izgradnjom projekta tako da pomoću definirane POM (*Project Object Model*) datoteke specificira na koji je način potrebno izgraditi projekt i koje sve biblioteke uključiti u izvršnu datoteku.

Nakon dohvaćanja izvornog koda upravljačkog uređaja možemo primijetiti složenu strukturu kazala koju prikazuje slika 6. Preporučeno je napraviti istu strukturu kazala koju imaju i jezgreni paketi, a to je najjednostavnije sljedećom naredbom (programski odsječak 2) iz korijenskog kazala upravljačkog uređaja:

**Programski odsječak 2** Kreiranje strukture kazala izvornog koda

```
1: $ mvn archetype:generate
        -DgroupId=com.example
        -DartifactId=novo-prosirenje
        -DarchetypeArtifactId=maven-archetype-quickstart
        -Dpackage=com.example.novo-prosirenje
        -DinteractiveMode=false
```

Nakon izvršavanja te naredbe imamo sljedeću strukturu kazala (slika 6):




**Slika 6:** Nastala struktura kazala

Prije nego što možemo početi pisati izvorni kod novog proširenja, potrebno je definirati datoteku *pom.xml* koja sadrži popis svih zavisnosti unutar uređaja, oznaku grupe proširenja, oznaku artefakta (naziv komponente ili projekta) kao i oznaku verzije proširenja. Unutar te datoteke također se specificira na koji se način *OSGi* paket koji predstavlja proširenje uređaja treba izgraditi.

Nakon što smo definirali strukturu kazala te datoteku *pom.xml* koja definira željeni način izgradnje projekta, preostaje nam samo ostvarenje *Java* razreda koji će raditi željene zadaće, te onda možemo napraviti paket novog proširenja korištenjem sljedeće naredbe (programski odsječak 3):

**Programski odsječak 3** Izgradnja paketa upravljačkog uređaja *OpenDaylight*

```
1: $ mvn clean install
```

## 4.5. Osnovni razredi proširenja

Kako bismo ostvarili novu *OSGi* komponentu, potrebna su nam samo dva *Java* razreda:

- ***OSGi* aktivator**

  koji će registrirati novu komponentu unutar *OSGi* radnog okvira.

- **Java razred proširenja**

  unutar kojeg ostvarujemo željene funkcije proširenja.

Prilikom razvijanja proširenja potrebno je ostvariti metode *init*, *start*, *stop* i *destroy* u *Java* razredu proširenja.

### 4.5.1. Razred *OSGi* aktivatora

*OSGi* aktivator zapravo je predložak *Java* razreda koji je potrebno nadopuniti. Unutar *Java* razreda proširujemo bazni razred *ComponentActivatorAbstractBase* upravljačkog uređaja. Taj razred ostvaruje metode *init*, *start*, *stop* i *destroy* koje poziva *OSGi* radni okvir prilikom pokretanja ili zaustavljanja proširenja upravljačkog uređaja. *Start* i *stop* metode upravljaju životnim ciklusom *OpenDaylight* proširenja, a one pozivaju dvije metode, *getImplementations* i *configureInstance*, koje je potrebno ostvariti unutar razreda *OSGi* aktivatora. Metoda *getImplementations* kao rezultat vraća *Java* razrede koji ostvaruju komponente novog proširenja. Jedno proširenje može ostvariti više od jedne komponente. Metoda *configureInstance* konfigurira proširenje te deklarira usluge koje pruža preko sučelja te usluge koje koristi.

Ostvarenje razreda *OSGi* aktivatora mora biti spremljeno pod nazivom *Activator.java* unutar kazala *novo-prosirenje/src/main/java/com/example/novo-prosirenje/*.

### 4.5.2. Java razred proširenja

Sam razred koji ostvaruje željenu logiku proširenja ima dva ograničenja. Razred mora ostvariti svaku od metoda *init*, *start*, *stop* i *destroy*, te se izvorni kod mora nalaziti unutar kazala *novo-prosirenje/src/main/java/com/example/novo-prosirenje/*.

Nakon što se ostvari željena logika i izgradi paket pomoću alata *Maven*, možemo pokrenuti upravljački uređaj *OpenDaylight* te pokrenuti novo proširenje.

## 4.6. Pokretanje proširenja

Pokretanje upravljačkog uređaja i zadanog proširenja odvija se u više koraka. Ako prije nismo, potrebno je izgraditi izvršnu datoteku upravljačkog uređaja *OpenDaylight*. Taj postupak dovoljno je napraviti samo jednom, najbolje nakon preuzimanja izvornog koda upravljačkog uređaja iz *git* repozitorija.

Sljedeći korak je izgradnja paketa s dodatnim proširenjem. Pozicioniramo se unutar kazala *novo-prosirenje/*, u kojem se nalazi datoteka *pom.xml*, te pokrenemo naredbu iz programskog odsječka 3. Alat *Maven* će nakon uspješno izvršene naredbe napraviti novo kazalo, *novo-prosirenje/target/*, unutar kojeg će se nalaziti paket proširenja s ekstenzijom *.jar*.

Pokretanje upravljačkog uređaja *OpenDaylight* radi se iz kazala *opendaylight/ distribution/ opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/* tako da se izvede sljedeća naredba (programski odsječak 4):

**Programski odsječak 4** Pokretanje upravljačkog uređaja *OpenDaylight*

```
1: $ ./run.sh -Xmx1G
```

Nakon pokretanja upravljačkog uređaja, dobivamo pristup *OSGi* konzoli unutar koje učitavamo novo proširenje. Unutar *OSGi* konzole potrebno je izvršiti sljedeće naredbe (programski odsječak 5):

**Programski odsječak 5** Pokretanje proširenja upravljačkog uređaja *OpenDaylight*

```
1: $ install file:///<put-do-prosirenja>/target/novo-prosirenje.jar
2: $ start <id-prosirenja>
```

Prvom naredbom učitamo proširenje u *OSGi* radni okvir. Kao rezultat izvršavanja te naredbe, ispisuju se podaci o učitanom paketu, između kojih je i *id-prosirenja*. To je brojčana oznaka koju koristimo za daljnje referenciranje na novo proširenje.

Nakon pokretanja naredbe *start <id-prosirenja>*, izvršava se metoda *start* iz *Java* razreda proširenja, čime smo zapravo pokrenuli novo proširenje.

# 5. Algoritam usmjeravanja zasnovanog na kvaliteti usluge

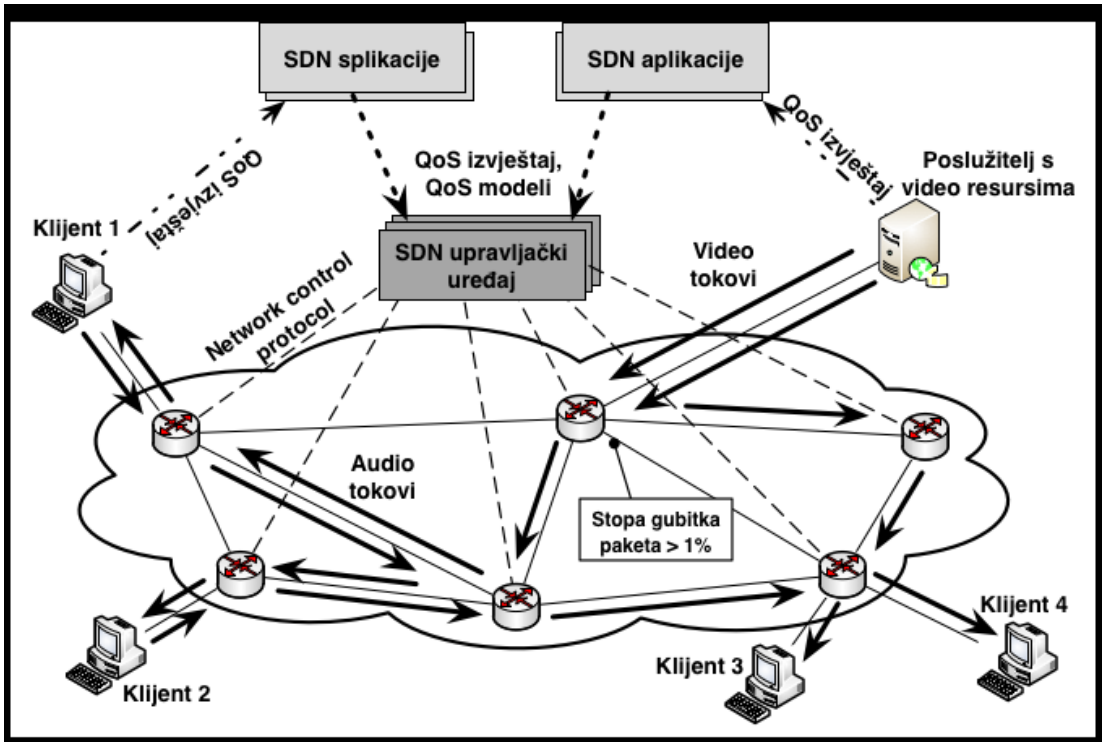
Ovo poglavlje predlaže algoritam usmjeravanja zasnovanog na kvaliteti usluge za SDN mreže te ostvarenje predloženog algoritma koristeći algoritam iz porodice heurističkih algoritama, optimizacijom kolonijom mrava. Ostvarenje algoritma izvedeno je u okviru proširenja upravljačkog uređaja *OpenDaylight* nazvanog *QoSRouting*.

## 5.1. Osnovni koncept algoritma usmjeravanja

Osnovni koncept algoritma usmjeravanja temelji se na usmjeravanju paketa s obzirom na kvalitetu usluge za različite vrste medija. SDN upravljački uređaj periodički prikuplja statističke podatke od mrežnih uređaja aktivnih u SDN mreži te na temelju prikupljenih podataka računa procjenu kašnjenja i procjenu stope gubitka paketa. Prilikom uključivanja ili isključivanja mrežnih uređaja u SDN mrežu, oni kontrolnim porukama dojavljuju svoju prisutnost odnosno odsutnost upravljačkom uređaju. Svaki mrežni uređaj periodički šalje upravljačkom uređaju i listu svojih neposrednih susjeda u SDN mreži. Na taj način on ima sve potrebne informacije o topologiji SDN mreže.

Upravljački uređaj, koji zna globalno stanje SDN mreže, prilikom zahtjeva klijenta prema poslužitelju, na temelju tražene vrste medija, pronalazi put koji najbolje odgovara specifičnim zahtjevima medija. Tako razlikujemo četiri vrste prometa, (a) video promet, (b) audio promet, (c) podatkovni promet i (d) ostali promet.

Slika 7 prikazuje osnovni koncept predloženog algoritma zasnovanog na kvaliteti usluge.




**Slika 7:** Osnovni koncept algoritma usmjeravanja

## 5.2. Optimizacija kolonijom mrava

Optimizacija kolonijom mrava (engl. *Ant Colony Optimization, ACO*) je algoritam koji po svojem načinu rada spada u kategoriju evolucijskih algoritama. Uže ga možemo svrstati među algoritme zasnovane na inteligenciji roja (engl. *swarm intelligence*), jer ne koristi metode specifične genetskim algoritmima već se zasniva na kolektivnom znanju i dijeljenju informacija među brojnim jedinkama.

Glavna ideja iz optimizacije kolonijom mrava je simulirati ponašanje mrava primjenom jednostavnih komunikacijskih mehanizama za rješavanje kompleksnih problema.

### 5.2.1. Ponašanje mrava u prirodi

Mrav koji se kreće u potrazi za hranom za sobom ostavlja tragove feromona. Drugi mravi koji osjete trag feromona prate taj trag do hrane te se vraćaju do mravinjaka. Ako negdje osjete jači trag feromona, veća je vjerojatnost da će krenuti tim putem.

Razmjena informacija putem feromona, tj. modifikacijom okoline se naziva **stigmergija**. Interakcija mrava je indirektna, tj. jedan mrav modificira okolinu, a drugi kasnije reagira na na novu okolinu, dok količina feromona na različitim putevima odre-

đuje vjerojatnost da će i drugi mravi slijediti taj put.

Što više mrava koristi neki put to će ostaviti više feromonskih tragova na tom putu. Feromoni s vremenom isparavaju, pa manje posjećeni putevi zapravo gube vjerojatnost odabira. Još jedan faktor koji utječe na snagu traga feromona je kvaliteta nalazišta hrane. Mravi će ostaviti više feromona ako je nalazište hrane veće i kvalitetnije.

Karakteristike ponašanja mrava ukazuju na decentraliziranu kontrolu koja zapravo pruža robusnost i fleksibilnost. Opisano ponašanje mrava može se izravno primijeniti na pakete i usmjeravanje unutar komunikacijske mreže.

### 5.2.2. Oponašanje mrava

Kako bi se ponašanje mrava ostvarilo na računalu, potrebno je definirati komponente algoritma poput:

- ponašanje mrava, tj. strategija konstrukcije rješenja,
- tragovi feromona,
- strategije osvježavanja feromona: isparavnje i pojačavanje,
- kriterij zaustavljanja,
- veličina populacije mrava.

Mravi konstruiraju rješenje s određenim pravilom prijelaza. Računalom simulirani mravi konstruktivno grade rješenja stohastičkim dodavanjem komponenti razmatrajući lokalne vrijednosti parametara umjetnih tragova feromona te samog stanja mrava (ograničenja problema i druge informacije heuristike specifične za problem).

Za dodatno ugađanje relativne važnosti tragova feromona i informacija heuristike specifične za problem uvode se parametri $\alpha$ i $\beta$. Kako bi bilo jasnije čemu točno služe, potrebno je najprije objasniti dva kriterija, diverzifikaciju i intenzifikaciju. Diverzifikacija odgovara istraživanju područja rješenja na način da se ravnomjerno pretraže sve regije prostora rješenja. Intenzifikacija pak odgovara iskorištavanju najboljih pronađenih rješenja, tj. detaljnje se pretražuje prostor rješenja koji će vjerojatnije imati bolje rezultate. Imajući to na umu, intenzifikaciju kod optimizacije kolonijom mrava postižemo povećanjem parametra $\alpha$, tj. dajemo veću težinu tragovima feromona. Diverzifikaciju postižemo povećanjem parametra $\beta$ za izbjegavanje slijeđenja već viđenih tragova.

Tragovi feromona opisuju trenutno stanje pretraživanja i predstavljaju memoriju cijelog procesa optimizacije kolonijom mrava. Model feromonskih tragova definira

vektor parametara modela $\tau$. Parametri odražavaju informacije relevantne za konstrukciju rješenja, pa tako parametar $\tau_{ij}$ predstavlja "interes" posjećivanja stanja $j$ iz stanja $i$.

Mravi konstruiraju rješenja prema probabilističkom pravilu prijelaza između stanja koje u obzir uzima vrijednosti feromona ($\tau_{ij}$) i informacije specifične za heuristiku ($\eta_{ij}$). Vjerojatnost odabira stanja $j$ iz stanja $i$ definirana je kao:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{k \in S} \tau_{ik}^{\alpha} \cdot \eta_{ik}^{\beta}}, \forall j \in S \tag{6}$$

pri čemu je $S$ skup neposjećenih rješenja.

Svaki mrav pojedinačno konstruira rješenje, te nakon što su svi mravi pronašli svoje rješenje, osvježavaju tragove feromona. Predloženi model koristi strategiju pojačavanja feromona zasnovanu na parametrima kvalitete. Pojačavaju se vrijednosti feromona svih pronađenih rješenja mrava s vrijednošću proporcionalnoj kvaliteti rješenja. Isparavanje tragova feromona ima ulogu prilagodljivog pamćenja rješenja jer se dinamički mijenja tijekom pretraživanja i izbjegava preranu konvergenciju algoritma. Feromonski trag automatski isparava sa stalnom stopom $\rho$, pri čemu vrijedi $\rho \in [0, 1]$. Isparavanje feromona se može definirati kao:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij}, \forall i, j \in [1, n] \tag{7}$$

pri čemu je $n$ broj čvorova zadane instance problema.

Kao kriterij zaustavljanja algoritma uzima se broj iteracija njegovog izvođenja. Veličina iteracije, tj. populacije mrava najčešće nije fiksna, već ovisi o veličini problema. Pseudokod korištenog algoritma za pronalaženje puta prikazan je u programskom odsječku 6.

**Programski odsječak 6** ACO pseudokod

```
1: Postavi parametre algoritma
2: Inicijaliziraj strukture podataka
3: procedure PRONADIPUT(izvor, odrediste)
4:     trenutniCvor = izvor
5:     while trenutniCvor ≠ odrediste do
6:         Odaberi sljedeći čvor j s vjerojatnošću p_{trenutniCvor,j}
7:         trenutniCvor = j
8:     end while
9: end procedure
10: procedure OPTIMIZACIJAKOLONIJOMMRAVA
11:     Inicijaliziraj tragove feromona
12:     while broj_iteracija > 0 do
13:         for each mrav do
14:             mrav = PronadiPut(izvor, odrediste)
15:         end for
16:         Primjeni ispravanje feromona
17:         Primjeni pojačavanje feromona
18:     end while
19:     return najbolje rješenje
20: end procedure
```

### 5.2.3. Model algoritma usmjeravanja

Problem usmjeravanja težinskog grafa rješavamo s mjerama kvalitete usluge kao ultimativnom metrikom. Budući da je problem pronalaska puta s višestrukim ograničenjima poznat kao NP-potpun problem i činjenica da je optimizacija kolonijom mrava pogodna tehnika za traženje puta na težinskom grafu, model algoritma usmjeravanja temelji se na optimizaciji kolonijom mrava koja pronalazi prihvatljiva rješenja za usmjeravnje temeljeno na kvaliteti usluge te postiže poboljšanja kvalitete usluge u usporedbi s usmjeravanjem na temelju najkraćeg puta. Težine bridova u grafu povezujemo s kašnjenjem i stopom gubitka paketa, koje dohvaća SDN upravljački uređaj. Dizajn heurističkog algoritma prikazan je u programskom odsječku 7. Optimizcija kolonijom mrava oponaša mrave, koji prelaze grafom prema zadaom odredištu, a nakon što pronađu put, ostavljaju tragove feromona na putu natrag prema izvorištu. Tragovi feromona stohastički utječu na rješenje tako da će čvorovi s jačim tragovima feromona

vjerojatnije biti dijelovi konačnog rješenja. Međutim, s vremenom tragovi feromona ishlapljuju, te se samo čvorovima koji su češće posjećeni od više različitih mrava povećava gustoća feromona. Budući da ovaj model pretpostavlja više različitih vrsta medija, za svaku vrstu medija postoji posebno programsko ostvarenje mrava, koji pokušava maksimizirati kvalitetu usluge za pripadajuću vrstu toka. Time je omogućeno lako dodavanje novih vrsta mrava za nove vrste medija i/ili tokova.

Šaljemo više mrava iste vrste iz istog izvorišta prema odredištu u nekoliko iteracija te se prilikom obilaska grafa čuva procjena kvalitete usluge kako bi se maksimizirao konačni rezultat. Prilikom obilaska grafa, svaki mrav čuva listu posjećenih čvorova. Kako bi se odredio sljedeći čvor kojeg će mrav posjetiti, za svakog mrava računa se vjerojatnost prijelaza za svaki susjedni čvor koji se ne nalazi na listi posjećenih čvorova prema izrazu 6.

Predloženi algoritam usmjeravanja sastoji se od više faza. Nakon definiranja parametara algoritma, $\alpha$, $\beta$, $populacijaMrava$, $brzinaIsparavanja$ i $brojIteracija$ (linija 1 u programskom odsječku 7), koji se mogu ugađati s obzirom na mrežno okruženje, prva faza je inicijalizacija tragova feromona i informacije specifične za heuristiku (linija 2). Informacija specifična za heuristiku za neki čvor u relaciji je s kašnjenjem i stopom gubitka paketa za taj čvor. Glavna logika algoritma nalazi se u funkciji $Kreni()$ (linija 13), koja, za zadanu vrstu medija, u $brojIteracija$ iteracija stvara populaciju mrava veličine $populacijaMrava$ te za svakog mrava traži put od izvorišta prema odredištu.

Kako bi skalarno izrazili kvalitetu usluge, moramo izračunati mjere kao što su kašnjenje (engl. *delay*) i gubitak paketa (engl. *packet loss*) s kraja na kraj. Budući da je kašnjenje aditivna mjera, kašnjenje s kraja na kraj računa se kao:

$$d_{e2e} = \sum_{cvor \in put} kasnjenje[cvor] \tag{8}$$

S druge strane gubitak paketa je multiplikativna mjera, te se stoga gubitak paketa s kraja na kraj računa kao:

$$l_{e2e} = 1 - \prod_{cvor \in put} (1 - stopaGubitka[cvor]) \tag{9}$$

Nakon što je svih $populacijaMrava$ mrava obišlo graf, (linije od 15 do 20), osvježavaju se vrijednosti feromona. Najprije, na svim se putevima ishlapljuje trag feromona (linija 21) te se zatim za svakog mrava koji je pronašao put od izvorišta do odredišta pojačava trag feromona s ostvarenom vrijednosti kvalitete usluge tog mrava (linije 24). Nakon što algoritam napravi $brojIteracija$ iteracija, put kojeg je pronašao mrav koji je ostvario najveću kvalitetu usluge vraća se kao konačno rješenje (linija 28).

**Programski odsječak 7** Model algoritma usmjeravanja temeljen na ACO algoritmu

```
1: Postavi parametre algoritma (α, β, populacijaMrava, brzinaIsparavanja, brojIteracija)
2: Inicijaliziraj tragove feromona (brojCvorova, kasnjenje, stopaGubitka)
3: procedure POSALJIMRAVA(mrav)
4:     trenutniCvor := izvorisniCvor
5:     mrav.dodajNaPut(trenutniCvor)
6:     while trenutniCvor ≠ odredisniCvor do
7:         vjerojatnosti := IzracunajVjerojatnosti(trenutniCvor)
8:         sljedeciCvor := IzaberiCvor(vjerojatnosti)
9:         mrav.dodajNaPut(sljedeciCvor)
10:        trenutniCvor := sljedeciCvor
11:    end while
12: end procedure
13: procedure KRENI(vrstaMedija)
14:    for k in 1..brojIteracija do
15:        for i in 1..populacijaMrava do
16:            mrav := StvoriMrava(vrstaMedija)
17:            PosaljiMrava(mrav)
18:            mrav.izracunajQoS()
19:            mravi.append(mrav)
20:        end for
21:        IshlapiFeromone()
22:        ukupnaVrijednostQoS := Σ_mrav mrav.dohvatiQoS()
23:        for mrav in mravi do
24:            PojacajFeromone(mrav, mrav.dohvatiQoS()/ukupnaVrijednostQoS)
25:        end for
26:    end for
27:    najboljiMrav := argmax_mrav (mrav.dohvatiQoS())
28:    return najboljiMrav.dohvatiPut()
29: end procedure
```

# 6. Programska izvedba predloženog algoritma

Za programsku izvedbu odabran je programski jezik *Java* jer je u istom programskom jeziku ostvaren i upravljački uređaj *OpenDaylight*. U poglavlju 4 opisan je postupak kreiranja novog proširenja upravljačkog uređaja *OpenDaylight*.

Za izradu proširenja, korištena je *Hydrogen* verzija upravljačkog uređaja *OpenDaylight*. Promjenu verzije upravljačkog uređaja radimo pomoću alata *git*, izvršavanjem sljedeće naredbe u korijenskom kazalu upravljačkog uređaja (programski odsječak 8).

**Programski odsječak 8** Promjena verzije upravljačkog uređaja *OpenDaylight*

```
1: $ git checkout stable/hydrogen
```

## 6.1. Proširenje *QoSRouting*

Ideja vodilja ovog zadatka bila je ostvariti reaktivno usmjeravanje paketa s obzirom na kvalitetu usluge. Način na koji je to zamišljeno je sljedeći: klijent uspostavlja sjednicu s glavnim poslužiteljem te u ovoj, pojednostavljenoj, inačici, poslužitelj odgovara s IP adresom poslužitelja koji posjeduje željeni resurs. Nakon toga klijent uspostavlja sjednicu s poslužiteljm te dohvaća željeni resurs.

Mrežni uređaji prosljeđuju pakete prema upravljačkom uređaju ukoliko ne pronađu odgovarajuće pravilo u vlastitoj tablici toka. Proširenje *QoSRouting* dohvaća proslјeđene pakete, te na temelju vrste i sadržaja paketa određuje sljedeću akciju.

Prilikom dolaska prvog paketa do upravljačkog uređaja, točnije do proširenja *QoSRouting*, stvara se novo pravilo usmjeravanja na mrežnim uređajima koji se nalaze na najkraćem putu između klijenta i glavnog poslužitelja. Traženje najkraćeg puta ostvareno je Dijkstrinim algoritmom. U trenutnu kad proširenje *QoSRouting* dobije paket

koji sadrži IP adresu poslužitelja sa željenim resursom, pokreće se postupak reaktivnog pronalaženja puta s obzirom na kvalitetu usluge koje se temelji na optimizaciji kolonijom mrava.

### 6.1.1. Funkcijski zahtjevi

S obzirom na zadatak, postavljeni su sljedeći funkcijski zahtjevi programske izvedbe:

- dohvaćanje i obrada paketa u upravljačkom uređaju
- izlučivanje informacija o odlaznim i dolaznim vratima i IP adresama
- izlučivanje sadržaja paketa transportnog sloja te rasčlanjivanje sadržaja paketa
- dinamično generiranje i instaliranje tokova u mrežnim uređajima
- pokretanje algoritma za usmjeravanje zasnovano na kvaliteti usluge

## 6.2. Povezivanje s drugim *OSGi* komponentama

Bitne informacije koje proširenje *QoSRouting* koristi za pronalazak puteva te evaluaciju kvalitete usluge dobiva od jezgrenih paketa preko sučelja koja su dostupna unutar *OSGi* radnog okvira (slika 8).

Sučelja na koja se *QoSRouting* oslanja su:

- *ISwitchManager*:

  služi za dohvat svih mrežnih uređaja, te brine oko ispada i dodavanja novih uređaja.

- *ITopologyManager*:

  služi za dohvat trenutne topologije komunikacijske mreže.

- *IStatisticsManager*:

  služi za dohvat statističkih podataka na temelju kojih se računaju procjene kašnjenja i gubitka paketa.

- *IForwardingRulesManager*:

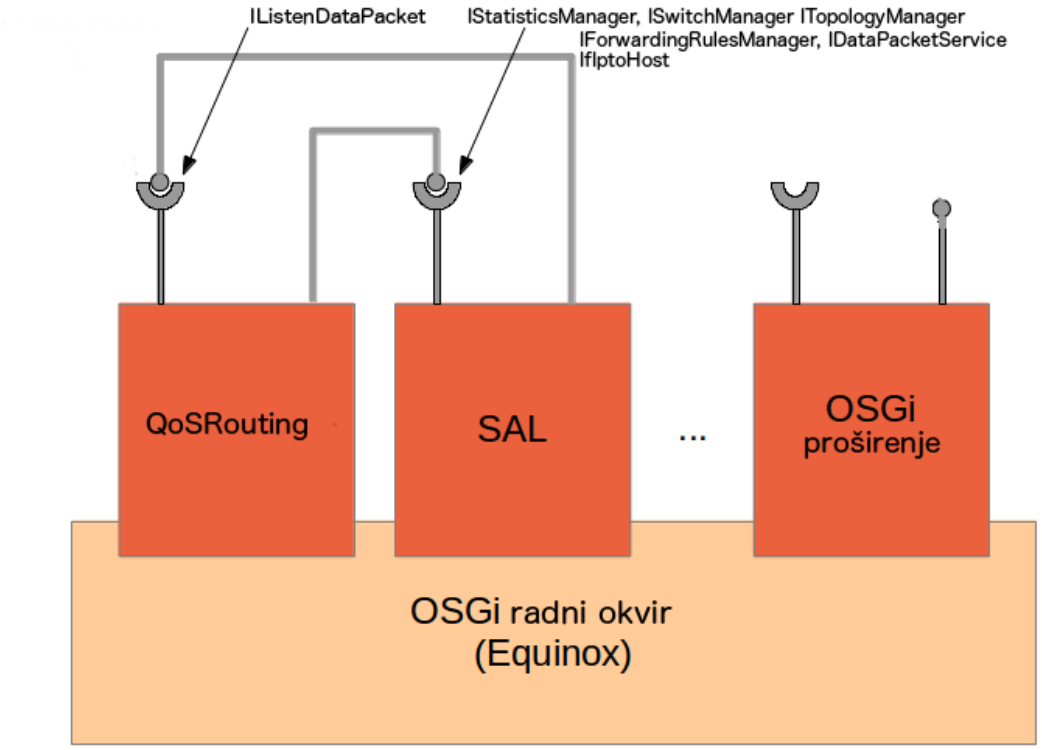
  omogućuje upravljanje nad pravilima u tablicama tokova mrežnih uređaja.

- *IDataPacketService*:

  pruža metode za enkapsulaciju/dekapsulaciju paketa svih slojeva OSI modela.

– *IfIptoHost*:

služi za dohvat IP adresa računala spojenih na mrežne uređaje.




**Slika 8:** Proširenje *QoSRouting* unutar *OSGi* radnog okvira

Samo povezivanje ostvaruje se u razredu *OSGi* aktivatora, *Activator.java*. U metodi *configureInstance* potrebno je navesti sva sučelja na koja će se proširenje oslanjati kako bi *OSGi* odredio poredak inicijalizacije komponenti. Dodatno, za svako od korištenih sučelja, u *Java* razredu proširenja potrebno je ostvariti metode *set* i *get*, koje će *OSGi* radni okvir pozvati prilikom inicijalizacije proširenja.

Primjer dodavanja jednog takvog sučelja, *ITopologyManager*, u razredu aktivatora prikazano je u nastavku (programski odsječak 9).

**Programski odsječak 9** Inicijalizacija sučelja unutar *OSGi* radnog okvira

```
1: c.add(createContainerServiceDependency(containerName)
2:         .setService(ITopologyManager.class)
3:         .setCallbacks("setTopologyManager", "unsetTopologyManager")
4:        .setRequired(true));
```

Dodatno, u datoteci *pom.xml* potrebno je navesti zavisnosti kako bi alat *Maven* pravilno izgradio paket proširenja (programsko proširenje 10).

**Programski odsječak 10** Deklaracija zavisnosti za alat *Maven*

```
<dependency>
    <groupId>org.opendaylight.controller</groupId>
    <artifactId>topologymanager</artifactId>
</dependency>
```

## 6.3. Dohvaćanje paketa u upravljačkom uređaju

Kako bi novo proširenje bilo u stanju dohvaćati pakete koji su namijenjeni upravljačkom uređaju, *Java* razred proširenja mora implementirati sučelje *IListenDataPacket* te nadjačati metodu *receiveDataPacket* (programski odsječak 11). Kao rezultat obrade, metoda mora vratiti jedan od sljedećih rezultata:

1. **CONSUME**

   Paket je obrađen te nema potrebe za daljnje prosljeđivanje drugim komponentama koje također slušaju na dolazak paketa.

2. **IGNORED**

   Paket je ignoriran pa ga je potrebno proslijediti sljedećoj komponenti koja sluša na dolazak paketa.

3. **KEEP_PROCESSING**

   Paket je obrađen ali ga svejedno treba proslijediti sljedećoj komponenti koja sluša na dolazaka paketa.

**Programski odsječak 11** Dohvaćanje paketa u upravljačkom uređaju

```
public class QoSRouting implements IListenDataPacket {
    ...
    @Override
    PacketResult receiveDataPacket(RawPacket inPkt) {
        ...
    }
    ...
}
```

### 6.3.1. Prepravka *Java* razreda *TCP.java*

Prilikom obrade dohvaćenih paketa, točnije prilikom obrade korisnog tereta (engl. *payload*) *TCP* paketa pojavljivali su se neočekivani rezultati. Detaljnom analizom obrade paketa, pronađena je pogreška u razredu *TCP.java* koji se nalazi u jednom od jezgrenih paketa *OpenDaylight*.

Uspoređivanjem metode koja je zadužena za odvajanje korisnog tereta i zaglavlja TCP segmenta te RFC 793 dokumenta, primjećena je razlika kod polja *Options* koje može biti varijabilne duljine dok se u implementaciji razreda *TCP.java* pretpostavljala fiksna duljina. Taj problem riješen je tako da se veličina *TCP* zaglavlja isčitala iz polja *Data Offset*. Nakon ponovne izgradnje upravljačkog uređaja taj se problem više nije pojavljivao.

## 6.4. Izračun procjene kašnjenja i gubitka paketa

Za evaluaciju kvalitete usluge potrebno je raspolagati procjenama kašnjenja i gubitka paketa. Sučelje *IStatisticsManager* periodički šalje upite mrežnim uređajima u kojima ih traži da mu pošalju statističke podatke, te nam po pozivu metode *getNodeConnector-Statistics* kao rezultat vraća prikupljene podatke. Međutim, sučelje *IStatistcisManager* nema podatke o kašnjenju.

### 6.4.1. Procjena kašnjenja

Za procjenu kašnjenja korištena je ideja iz [17], a temelji se na procjeni kašnjenja na temelju broja zapisa u tablici toka.

Pomoću sučelja *IForwardingRulesManager* možemo dohvatiti koliko zapisa u tablici toka sadrži svaki mrežni uređaj. Vrijednost koju uzimamo kao procjenu kašnjenja, dobiva se korištenjem sljedećeg izraza:

$$\hat{d} = q \cdot t_{MTU}, \tag{10}$$

pri čemu je $q$ broj zapisa u tablici toka, a $t_{MTU}$ vrijeme koje je potrebno da prenese najveći mogući paket (engl. *Maximum Transmission Unit, MTU*) koristeći vezu $100Mb/s$. Veličina *MTU* iznosi $1500okteta$ što rezultira vremenom prijenosa od $t_{MTU} = 120\mu s$.

Kako ne bismo smanjili općenitost, proširenje *QoSRouting* prilikom izračuna procjene kašnjenja dinamički određuje vrijednost $t_{MTU}$ na način da od mrežnog uređaja zatraži nazivnu vrijednost propusnosti umjesto gore pretpostavljene $100Mb/s$.

### 6.4.2. Procjena gubitka paketa

Procjena gubitka paketa računa se na temelju podataka o poslanim, odnosno primljenim paketima, koristeći sljedeći izraz:

$$\hat{p} = \frac{|primljeni\_paketi(i)|}{\sum_{j \in susjed(i)} |poslani\_paket(j,i)|},$$

pri čemu je $|primljeni\_paketi(i)|$ broj paketa koje je primio mrežni uređaj $i$, dok je $|poslani\_paket(j,i)|$ broj paketa koje je mrežni uređaj $j$ poslao prema mrežnom uređaju $i$. Upravljački uređaj *OpenDaylight*, koristeći specifikaciju *OpenFlow*, periodički prikuplja statističke podatke od mrežnih uređaja, između kojih je i broj primljenih odnosno poslanih paketa.

## 6.5. Traženje puteva

Jedan od ključnih dijelova zadatka je traženje puteva. Tijekom razvijanja proširenja *QoSRouting*, posebna pažnja stavljena je na modularnost komponenta za traženje puteva kako bi se ostvarila platforma za ispitivanje različitih izvedbi algoritama.

Modul koji je zadužen za pronalazak puteva pronađeni put mora vratiti u obliku liste bridova (programski odsječak 12).

**Programski odsječak 12** Primjer pozivanja modula za traženje puteva

```
1: List<Edge> path;
2: path = Dijkstra.getPathHopByHop();
3: // ili
4: path = ACO.getPath();
```

Razred *Edge* je jedan od jezgrenih *Java* razreda upravljačkog uređaja *OpenDaylight*.

### 6.5.1. Dijkstrin algoritam

Dijkstrin algoritam jedan je od poznatijih algoritama u kominikacijskim mrežama te se ovdje neće ulaziti u detaljan opis. Njegovo ostvarenje unutar proširenja *QoSRouting* nalazi se u razredu *Dijkstra.java*, te je napravljeno zbog usporedbe s drugim algoritmima koji svoj rad temelje na kvaliteti usluge.

### 6.5.2. Optimizacija kolonijom mrava

Optimizacija kolonijom mrava detaljnije je opisana u poglavlju 5.2, a ovdje ćemo dati više implementacijske detalje.

Glavni razred zadužen za optimizaciju kolonijom mrava je *AntColony.java*. U njemu je ostvaren pseudokod iz programskog odsječka 6, u programskom jeziku *Java*.

Za svaki od tri različite vrste medija, ostvaren je poseban razred "mrava". Svaki razred implementira bazni razred *Ant.java* nad kojim onda nadjačava metode za izračun kvalitete usluge prema vrsti medija. Tako postoji razred mrava za video promet pod nazivom *AntVideo.java*, za audio promet *AntVoice.java* te za podatkovni promet *AntData.java*.

Dodatno, sam konstruktor razreda *AntColony.java* zahtijeva dodatne parametre koje Dijkstrin algoritam ne zahtijeva, a to su procjene kašnjenja te gubitka paketa.

## 6.6. Stvaranje novih pravila usmjeravanja

Nakon što smo dobili paket na upravljačkom uređaju, odredili procjene kašnjenja te gubitka paketa, pronašli put između izvora i odredišta, preostaje još samo kreirati nova pravila usmjeravanja za mrežne uređaje koji se nalaze na pronađenom putu.

Prilikom primitka paketa, najprije se iz njega izdvoje izvorišna i odredišna adresa te transportna vrata. Naime, kako bismo omogućili pronalaženje puteva koji potencijalo pružaju bolju kvalitetu usluge pri opterećenim mrežama, svaki tok je izravno vezan za izvor i odredište te korištena vrata.

Za svaki mrežni uređaj na pronađenom putu stvaraju se dva unosa u tablici toka. Jedan je za smjer od izvora prema odredištu, a drugi je za smjer od odredišta prema izvorištu. Svaki od unosa sadrži sljedeća polja za "prepoznavanje" paketa:

**Programski odsječak 13** Stvaranje pravila usmjeravanja - *match*

```
1: match.setField(MatchType.DL_TYPE, EtherTypes.IPv4.shortValue());
2: match.setField(MatchType.NW_DST, dstAddr);
3: match.setField(MatchType.NW_SRC, srcAddr);
4: match.setField(MatchType.NW_PROTO, transportProto);
5: match.setField(MatchType.TP_SRC, srcPort);
6: match.setField(MatchType.TP_DST, dstPort);
```

Akcija koju mrežni uređaj mora napraviti nailaskom paketa koji odgovara unosu, definira se kao:

**Programski odsječak 14** Stvaranje pravila usmjeravanja - *action*

```
1: actions.add(new Output(nodeConnector));
```

Time definiramo da mrežni uređaj dobiveni paket mora staviti na izlaz koji je označen s $nodeConnector$. Tako definirana pravila prosljeđivanja šaljemo na mrežne uređaje pomoću sučelja *IForwardingRulesManager* (programski odsječak 15).

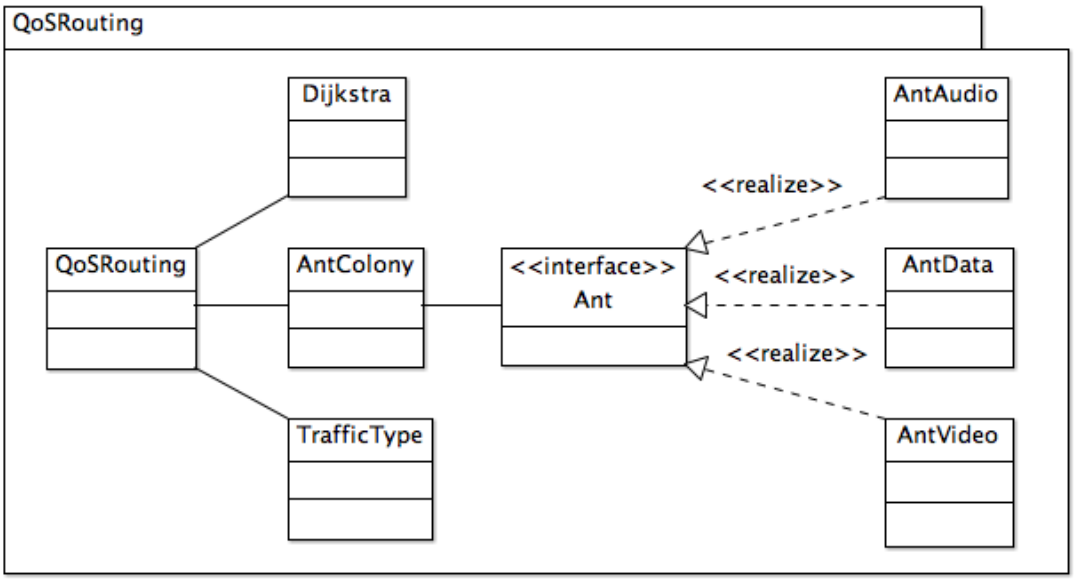
**Programski odsječak 15** Stvaranje pravila prosljeđivanja

```
1: this.forwardRulesManager.installFlowEntry(flowEntry);
```

## 6.7. Dijagrami razreda proširenja

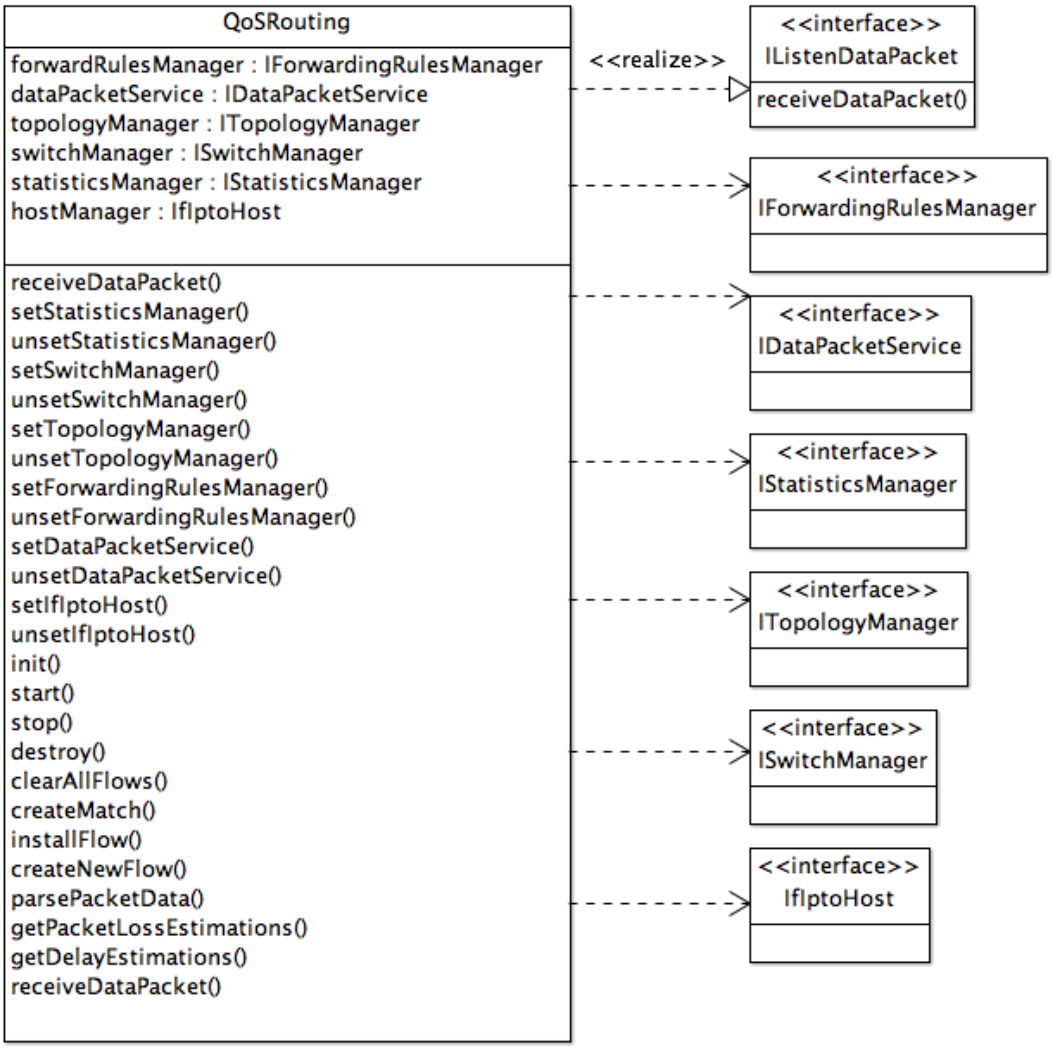
U nastavku su prikazani dijagrami razreda ostvarenog proširenja.

Na slici 9 je prikazan pojednostavljen dijagram razreda proširenja *QoSRouting*. Ono se sastoji od sljedećih *Java* razreda: *QoSRouting.java*, *Dijkstra.java*, *AntColony.java*, *TrafficType.java*, *AntAudio.java*, *AntVideo.java*, *AntData.java* te sučelja *Ant.java*. *Java* razred *TrafficType.java* sadrži vrste medija koje proširenje razlikuje.



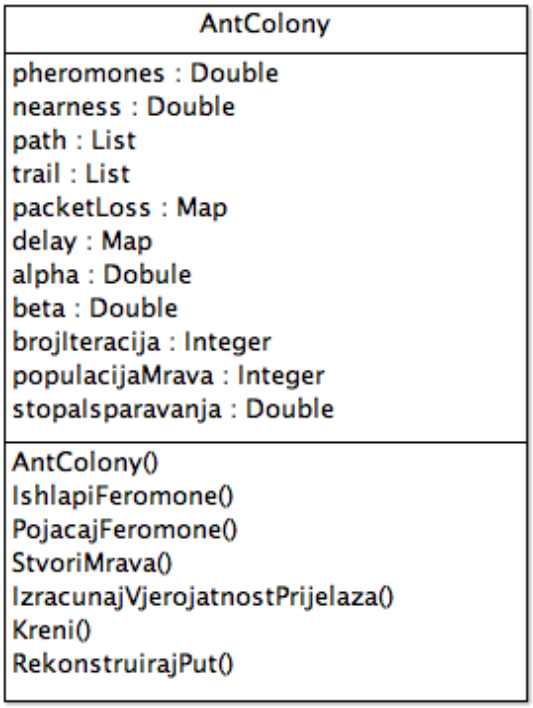

**Slika 9:** Dijagram razreda proširenja *QoSRouting*

Na slici 10 detaljnije je prikazano ostvarenje *Java* razreda *QoSRouting.java*. Možemo primijetiti sučelja koja su potrebna kako bi od upravljačkog uređaja *OpenDaylight* dobili informacije o topologiji, statističke podatke, mrežne uređaje te imali mogućnost dohvata nadolazećih paketa.



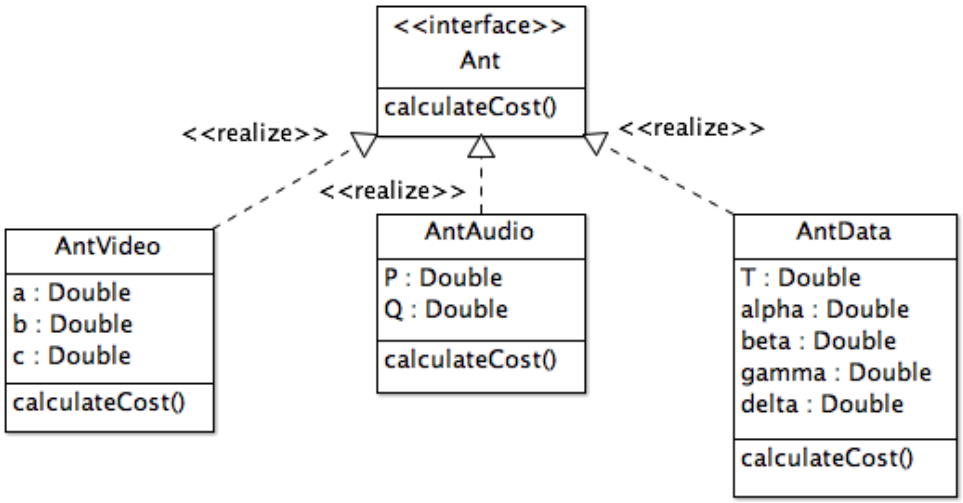

**Slika 10:** Dijagram razreda *QoSRouting Java* razreda

Na slici 11 prikazano je ostvarenje *Java* razreda *AntColony.java* u kojoj je ostvarena optimizacija kolonijom mrava opisana u programskom odsječku 7.




**Slika 11:** Dijagram razreda *AntColony Java* razreda

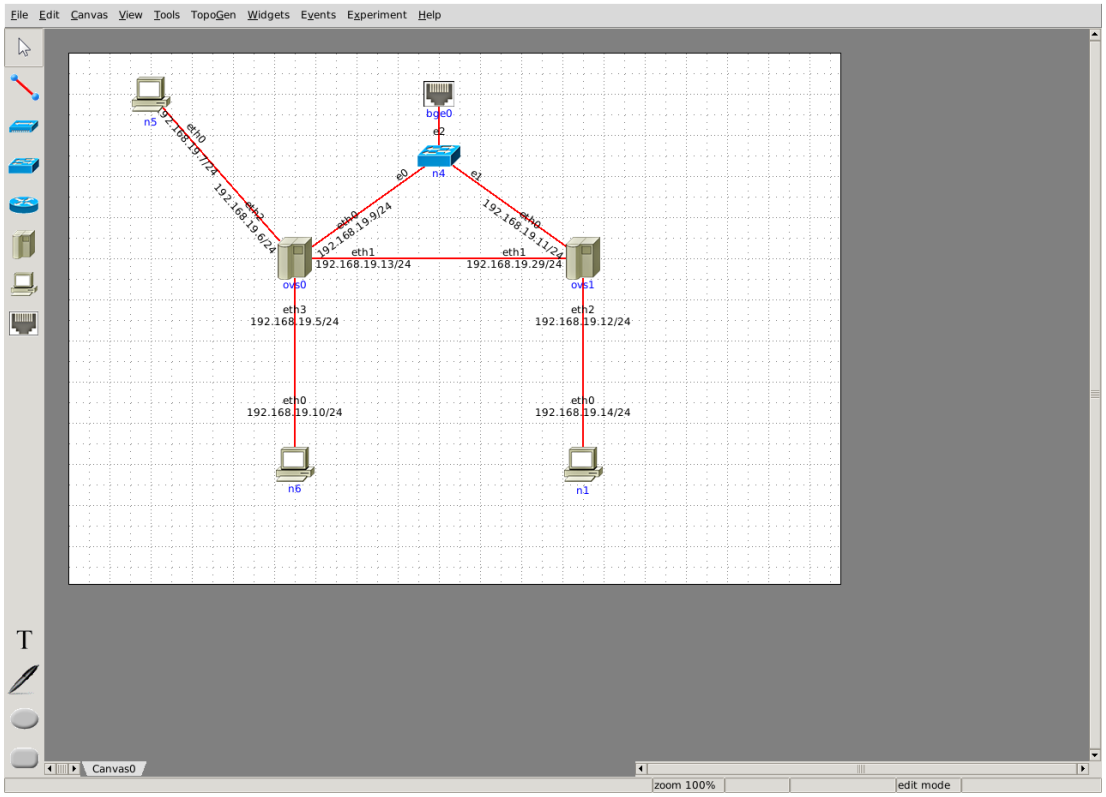
Na slici 12 prikazana su različita ostvarenja mrava za pojedine vrste medija koje algoritam podržava.




**Slika 12:** Dijagram razreda *Ant Java* razreda

## 6.8. Demonstracija rada

Rad proširenja *QoSRouting* je dodatno ispitan i pomoću simulatora/emulatora IMUNES.




**Slika 13:** Topologija mreže u simulatoru/emulatoru IMUNES

Na jednom računalu instaliran je i podešen upravljački uređaj OpenDaylight. Drugo računalo, fizički odvojeno od prethodno spomenutog računala, koristi se za pokretanje simulatora/emulatora IMUNES. To računalo ima dva dodatna mrežna sučelja. Jedno od tih sučelja koristi se za spajanje komunikacijske mreže u simulatoru/emulatoru IMUNES s upravljačkim uređajem koji je pokrenut na drugom računalu.

### 6.8.1. Povezivanje simulatora/emulatora IMUNES i upravljačkog uređaja OpenDaylight

Upravljački uređaj OpenDaylight i komunikacijska mreža unutar simulatora/emulatora IMUNES, prema OpenFlow specifikaciji, mogu komunicirati ukoliko se nakon pokretanja IMUNES eksperimenta, pokrene skripta *netinit2.py* iz [16]. Unutar te skripte nalaze se naredbe koje je potrebno pokrenuti na svakom mrežnom komutatoru unutar simulatora/emulatora IMUNES kako bi se podigla usluga zadužena za razmjenu

poruka s upravljačkim uređajem *OpenDaylight*, *Open vSwitch*. Nakon pokretanja te skripte, upravljački uređaj *OpenDaylight* i mrežni komutatori mogu razmjenjivati poruke definirane *OpenFlow* specifikacijom. Pokretanje skripte se radi na sljedeći način:

**Programski odsječak 16** Povezivanje simulatora/emulatora IMUNES i upravljačkog uređaja OpenDaylight

```
1: python netinit2.py –c 192.168.19.116:6633 –s 192.168.19.255
2: python netinit2.py -kill
```

U gore navedenom primjeru, adresa upravljačkog uređaja te maska podmreže (engl. *Subnet Mask*) odgovaraju onima sa slike 13. Prva se naredba koristi nakon što pokrenemo eksperiment unutar IMUNES-a, a ona pokreće uslugu *ovs-switch*, koja zapravo omogućuje komutatoru da komunicira s upravljačkim uređajem koristeći specifikaciju OpenFlow, na svim komutatorima. Druga se naredba koristi prije gašenja eksperimenta, a ona osigurava da se na svim komutatorima pravilno ugasi *ovs-switch* uslugu.

# 7. Evaluacija predloženog algoritma

Za potrebe mjerenja rezultata napravljena je promjena proširenja *QoSRouting*. Kako bi svi algoritmi imali istu polazišnu točku, metode koje računaju procjene kašnjenja i gubitka paketa uvijek vraćaju iste rezultate za kašnjenje te gubitak paketa, bez obzira na trenutno pravo stanje mreže. Time zapravo dajemo svim algoritmima iste početne uvjete tako da možemo uspoređivati njihove rezultate. Uz to, u metode za traženje puteva dodan je ispis vremena izvođenja kao i vrijednosti kvalitete usluge.

Za potrebe evaluacije korišteni su modeli procjene kvalitete usluge iz [2]. Ti modeli izražavaju kvalitetu usluge na MOS (*Mean Opinion Scale*) skali od 1 (najgora kvaliteta) do 5 (najbolja kvaliteta).

Model za izračun kvalitete usluge kod audio prometa:

$$QoE_{audio} = T - \gamma \cdot l_{e2e} + \delta \cdot d_{e2e} - \epsilon \cdot (d_{e2e})^2 + \zeta \cdot (d_{e2e})^3,$$

pri čemu je $T$ gornja granica vrijednosti kvalitete usluge kada u mreži nisu prisutni gubici paketa niti kašnjenje, dok su $\gamma$, $\delta$, $\epsilon$ i $\zeta$ vrijednosti specifične za model a gornja granica kvalitete usluge iznosi $4.3$.

Model za izračun kvalitete usluge kod video prometa:

$$QoE_{video} = 1 + R(c_f, o_f) \cdot \exp(-\frac{l_{e2e}}{Q(c_f)}),$$

pri čemu su funkcije $R$ i $F$ funkcije specifične za vrstu video kodeka ($c_f$) i brzine prijenosa ($o_f$), a za gornju granicu vrijednost kvalitete usluge se uzima iznos $4.53$.

Model za izračun kvalitete usluge kod podatkovnog prometa:

$$QoE_{data} = \lambda \cdot \log(\mu \cdot o_f \cdot (1 - l_{e2e})),$$

pri čemu je $o_f$ prosječna brzina prijenosa, $\lambda$ i $\mu$ su konstante specifične za model, dok je vrijednost gornje granice kvalitete usluge $4.5$.

## 7.1. Laboratorijsko okruženje

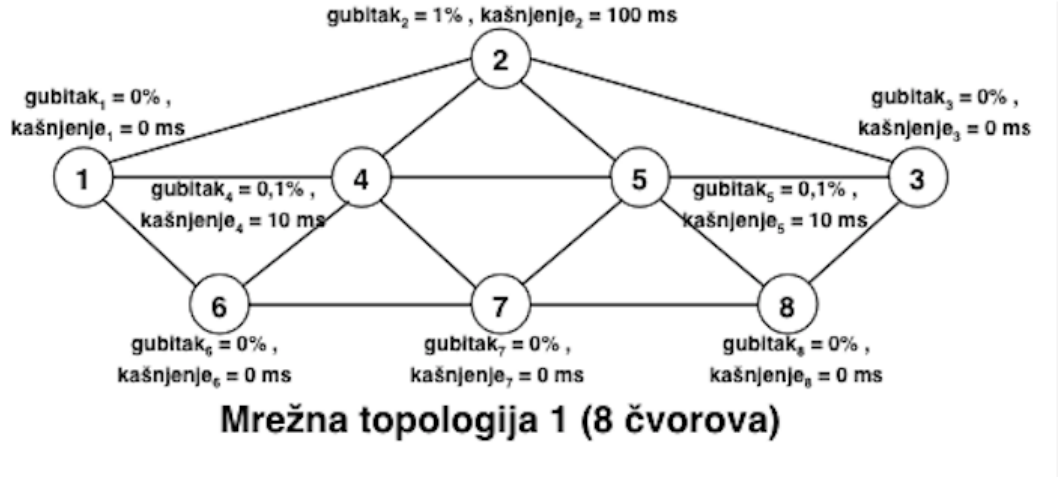
Proširenje *QoSRouting* razvijano je i testirano u laboratorijskom okruženju na Fakultetu elektrotehnike i računarstva, na Zavodu za telekomunikacije. Korišteno je laboratorijsko računalo sljedećih tehničkih značajki:

- **CPU**: Intel Core i3, $4 \times 3.20$ GHz
- **RAM**: 4 GB
- **HDD**: 50 GB
- **OS**: GNU/Linux Ubuntu 12.04.5 LTS
- **Emulator mreže**: mininet 2.0.0

Za sva se mjerenja koristio alat *iperf* koji vrši isptivanja propusnosti mreže. Za potrebe mjerenja razvijena je *Python* skripta koja radi sljedeće: u sklopu emulatora mreže *mininet* definira topologiju komunikacijske mreže. Nakon toga spaja sve mrežne uređaje na upravljački uređaj koji mora biti pokrenut, te započinje s nizom preddefiniranih *iperf* naredbi. Svaka *iperf* naredba definira par izvorište, odredište, vrstu prometa, transportna vrata ta željenu propusnost. Transportna vrata određuju vrstu medija prema kojoj se vrši evaluacija kvalitete usluge. Nakon svake naredbe ispisuju se rezultati koji su prikazani u nastavku.

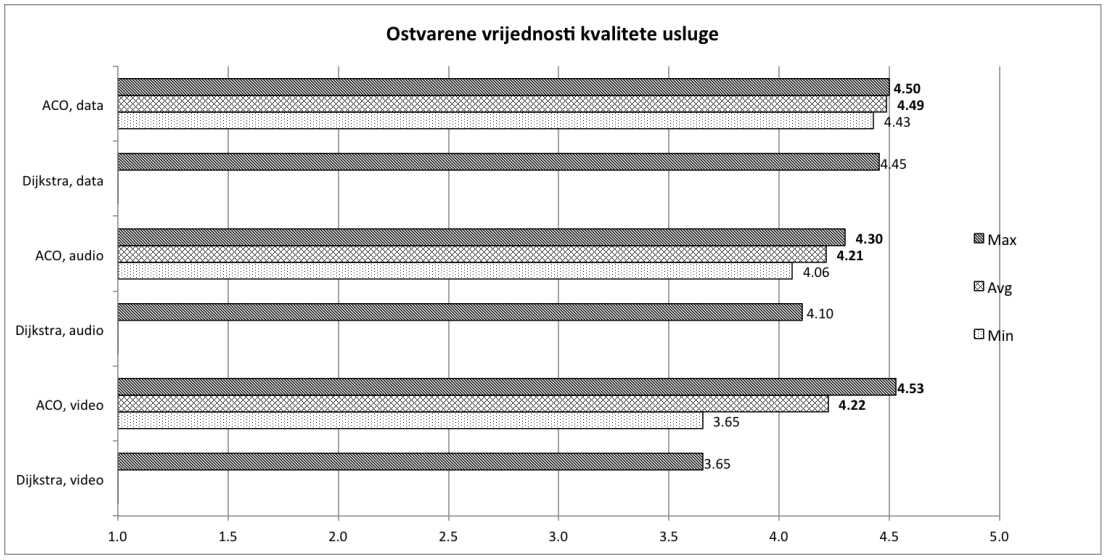
## 7.2. Mjerenja

Na slici 14 prikazana je topologija mrežnih uređaja korištena za prvo mjerenje. Kraj svakog mrežnog uređaja stoji nazivna vrijednost kašnjenja i gubitka paketa koja je korištena. Krajnja računala, koja su inicirala promet, bila su spojena na mrežne uređaje s oznakom 1 i 3.




**Slika 14:** Topologija 1. SDN mreže za evaluaciju

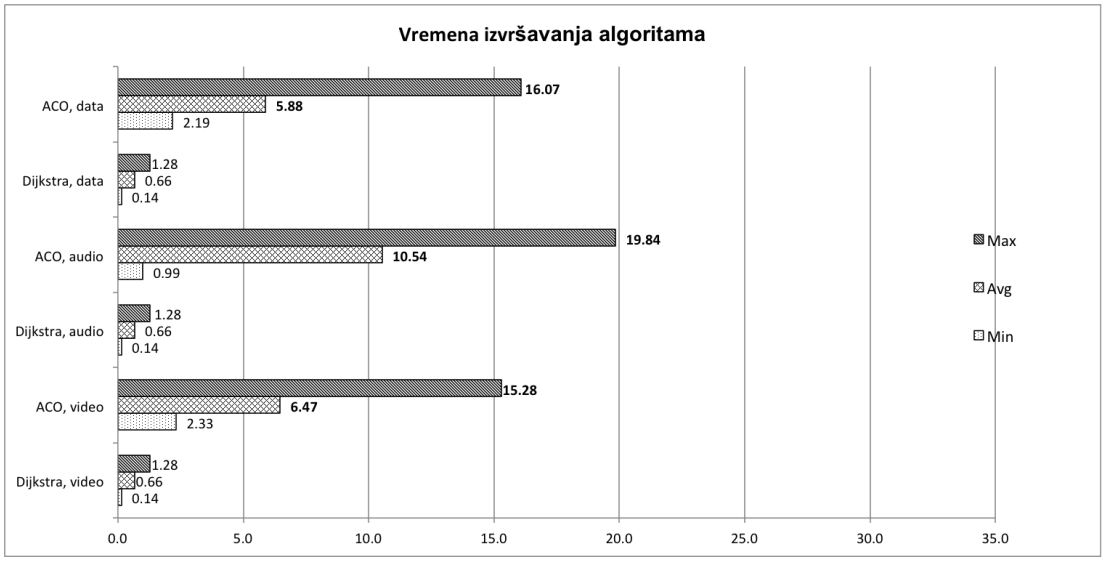
Na sljedećem grafu (slika 15) prikazana je usporedba ostvarenih vrijednosti kvalitete usluge za različite algoritme izraženog na *MOS* skali od 1 do 5.




**Slika 15:** Usporedba ostvarenih vrijednosti kvalitete usluge (topologija 1)

Možemo primijetiti da je za svaku vrstu prometa vrijednost kvalitete usluge kod puteva pronađenih koristeći optimizaciju kolonijom mrava uvijek veća ili jednaka onoj kod puteva pronađenih Dijkstrinim algoritmom.

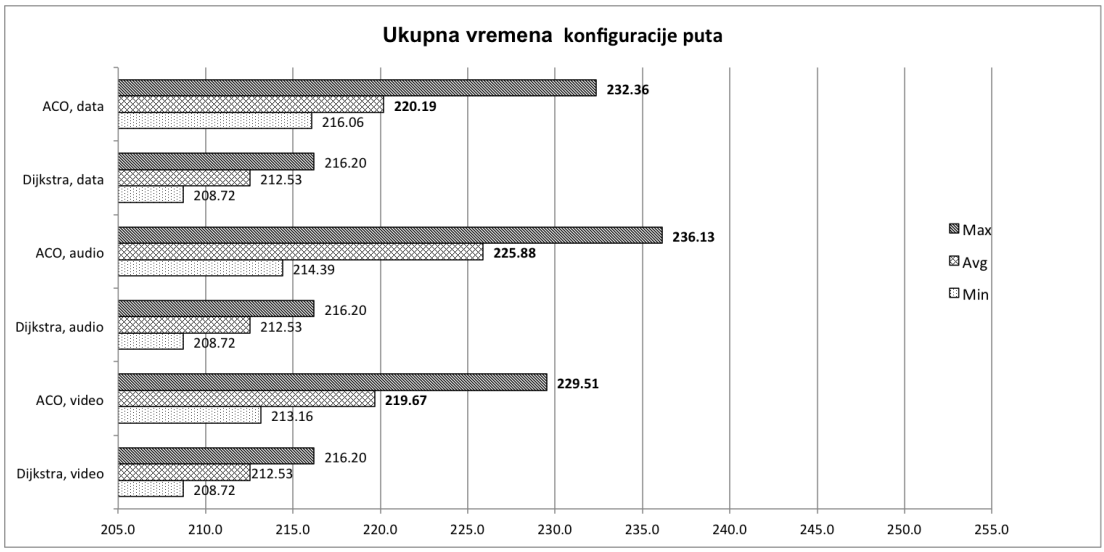
Na sljedećem grafu (slika 16) prikazana je usporedba vremena izvršavanja algoritama za traženje puta izraženog u milisekundama.




**Slika 16:** Usporedba vremena izvršavanja algoritama za traženje puta (topologija 1)

Možemo primijetiti da se trajanje izvršavanja optimizacije kolonijom mrava izvršava višestruko dulje od Dijkstrinog algoritma.

Na sljedećem grafu (slika 17) je prikazana usporedba vremena konfiguracije puta izražnog u milisekundama, od dolaska paketa do stvaranja novih pravila usmjeravanja.

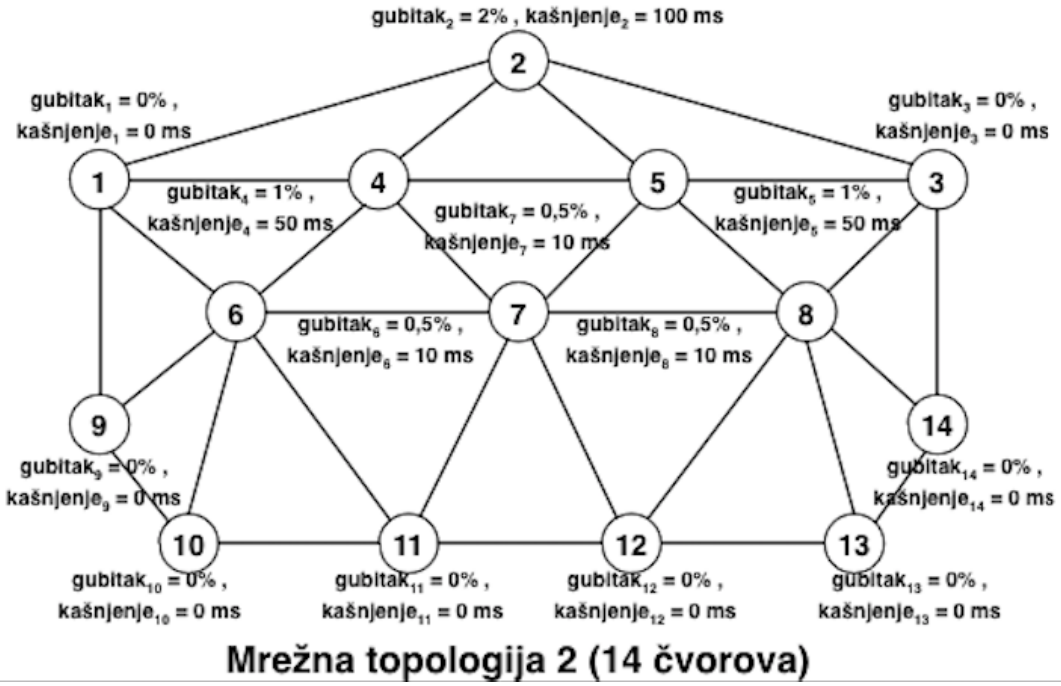



**Slika 17:** Usporedba ukupnog vremena konfiguracije puta (topologija 1)

Možemo primijetiti da se kod ukupnog vremena izvršavanja smanjila razlika između trajanja optimizacije kolonijom mrava i Dijkstrinog algoritma.

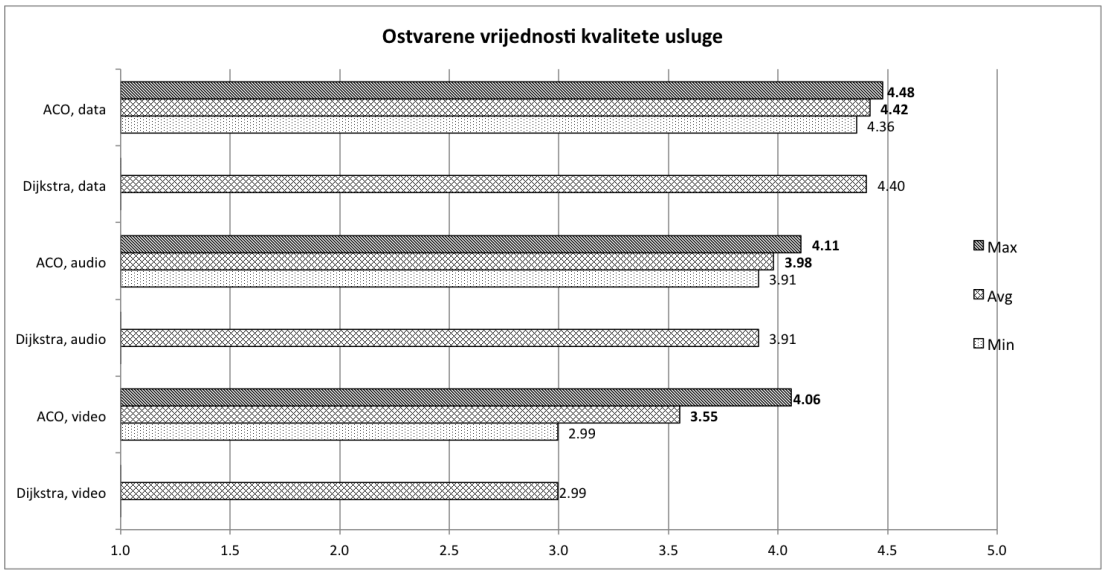
Na slici 18 prikazana je topologija mrežnih uređaja korištena za drugo mjerenje. Kraj svakog mrežnog uređaja stoji nazivna vrijednost kašnjenja i gubitka paketa koja

je korištena. Krajnja računala, koja su inicirala promet, bila su spojena na mrežne uređaje s oznakom 1 i 3.




**Slika 18:** Topologija 2. SDN mreže za evaluaciju

Na sljedećem grafu (slika 19) prikazana je usporedba ostvarenih vrijednosti kvalitete usluge za različite algoritme izraženog na *MOS* skali od 1 do 5.

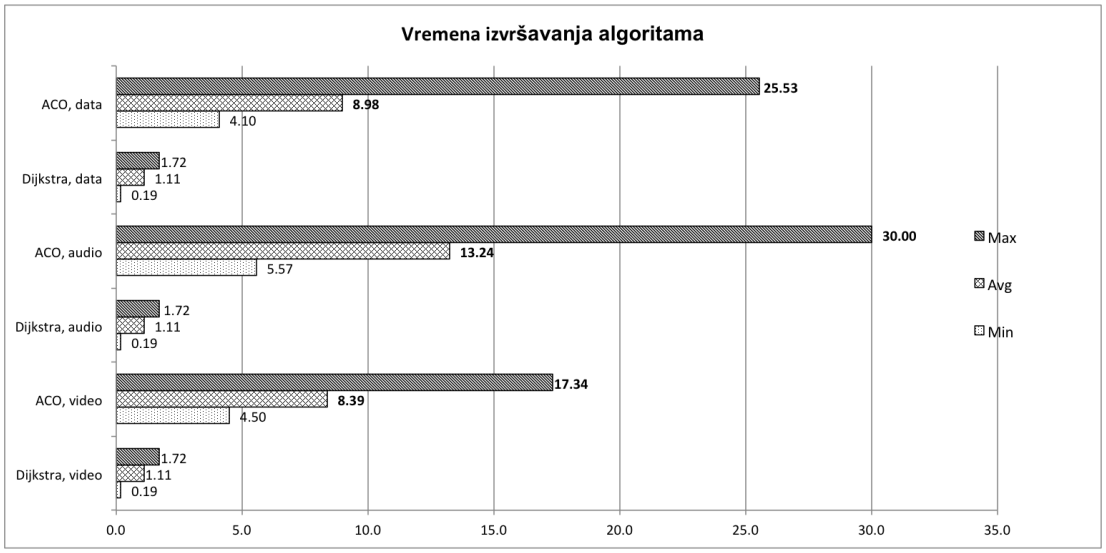



**Slika 19:** Usporedba ostvarenih vrijednosti kvalitete usluge (topologija 2)

Možemo primijetiti da je za audio i video promet, vrijednost kvalitete usluge kod

puteva proneđenih koristeći optimizaciju kolonijom rmava uvijek veća ili jednaka onoj kod puteva pronađenih Dijkstrinim algoritmom. Jedina razlika s obzirom na rezultate sa slike 15 je što je najmanja izmjerena vrijednost kvalitete usluge za podatkovni promet ispod izmjerene vrijednosti za Dijkstrin algoritam.

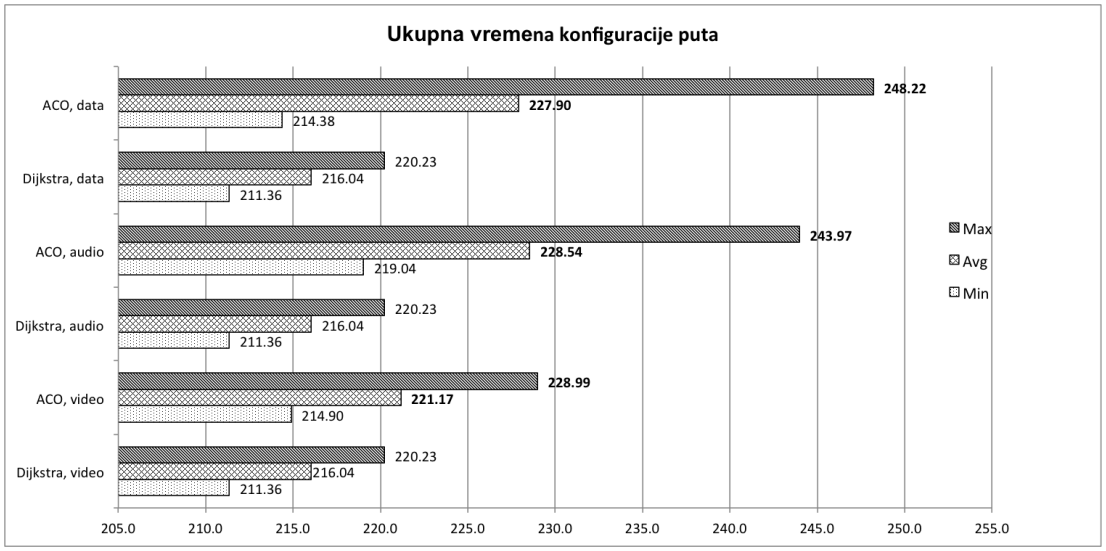
Na sljedećem grafu (slika 20) prikazana je usporedba vremena izvršavanja algoritama za traženje puta izraženog u milisekundama.




**Slika 20:** Usporedba vremena izvršavanja algoritama za traženje puta (topologija 2)

S povećanjem topologije može se očekivati i povećanje vremena potrebnog za izvršavanje algoritama za traženje puteva, pa su tako porasla vremena za sve vrste prometa s obzirom na rezultate sa slike 16. Razlika između trajanja optimizacije kolonijom mrava i Dijkstrinog algoritma i dalje iznosi jedan red veličine.

Na sljedećem grafu (slika 21) je prikazana usporedba vremena konfiguracije puta izražnog u milisekundama, od dolaska paketa do stvaranja novih pravila usmjeravanja.




**Slika 21:** Usporedba ukupnog vremena konfiguracije puta (topologija 2)

Ukupna vremena izvršavanja su porasla u odnosu na rezultate sa slike 17, no možemo primijetiti da je razlika između trajanja optimizacije kolonijom mrava i Dijkstrinog algoritma i dalje mala.

# 8. Zaključak

Zadatak ovog rada bio je proučiti koncept programski upravljanih mreža, specifikaciju *OpenFlow*, upravljački uređaj *OpenDaylight* te postojeće algoritme za usmjeravanje koje je zasnovano na kvaliteti usluge, kako bi se ostvarilo proširenje nad upravljačkim uređajem za usmjeravanje zasnovano na kvaliteti usluge, točnije, implementacija heurističkog algoritma zasnovanog na optimizaciji kolonijom mrava te modelima kvalitete usluge.

U sklopu rješenja zadatka napravljeno je proširenje *QoSRouting* u sklopu upravljačog uređaja *OpenDaylight* koje dohvaća pakete usmjerene na upravljački uređaj, pronalazi put između izvorišta i odredišta te stvara nova pravila usmjeravanja kojima onda konfigurira mrežne uređaje. Traženje puteva izvedeno je modularno kako bi se što lakše omogućilo ispitivanje različitih algoritama za usmjeravanje zasnovano na kvaliteti usluge.

Dobiveni rezultati pokazuju poboljšanje kvalitete usluge predloženog algoritma upotrebom usmjeravanja zasnovanog na kvaliteti usluge u odnosu na klasično usmjeravanje najkraćim putem. Usporednom vremena izvršavanja vidljivo je da je klasično usmjeravanje Dijkstrinim algoritmom neznatno brže, a usporedba ukupnih vremenena konfiguracije puta pokazuje da ono iznosi najviše $4.47\%$ odnosno $5.99\%$ od ukupnog vremena konfiguracije puta.

Ostvarena kvaliteta usluge predloženog algoritma bolja je od kvalitete usluge ostvarene usmjeravanjem Dijkstrinim algoritmom, dok je ostvareno vrijeme usmjeravanja Dijkstrinim algoritmom bolje od ostvarenog vremena predloženog algoritma. Dobiveni rezultati pokazuju isplativost korištenja predloženog algoritma nad usmjeravanjem Dijkstrinim algoritmom zbog dobivene veće kvalitete usluge za cijenu neznatno većeg vremena izvršavanja.

# LITERATURA


[1] S. Agarwal, M. Kodialam, i T. V. Lakshman. Traffic Engineering in Software Defined Networks. *IEEE Conferenceon Computer Communications*, stranice 2211–2219, 2013.

[2] O. Dobrijević, A. J. Kassler, L. Skorin-Kapov, i M. Matijašević. Q-POINT: QoE-Driven Path Optimizaion Model for Multimedia Services. *Wired/Wireless Internet Communications, ser. Lecture Notes in Computer Science. Springer International Publishing*, 8458:134–147, 2014.

[3] H. E. Egilmez i A. M. Tekalp. Distributed QoS Architectures for Multimedia Streaming Over Software Defined Networks. *IEEE Transactions on Multimedia*, 16(6):1597–1609, 2014.

[4] H. E. Egilmez, S. Civanlar, i A. M. Tekalp. An Optimization Framework for QoS-Enabled Adaptive Video Streaming Over OpenFlow Networks. *IEEE Transactions on Multimedia*, 15(3):710–715, 2013.

[5] A. Iwata et al. ATM routing algorithms with multiple qos requirements for multimedia internet-working. *IEICE Transactions and Communications*, E79-B(8): 999–1006, 1996.

[6] Open Networking Foundation. Openflow switch specification v.1.4.0. `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf`, 2013. ONF White Paper.

[7] L. Gang i K. G. Ramakrishnan. A*prune: an algorithm for finding K shortest paths subject to multiple constraints. *The 20th Annual Joint Conference of the IEEE Computer and Communications Societes*, stranice 743–749, 2001.

[8] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, i N. Race. Towards Network-wide QoE Fairness Using OpenFlow-assisted Adaptive Video Stre-

aming. *ACM SIGCOMM Workshop onFuture Human-centric Multimedia Networking*, stranice 15–20, 2013.

[9] J. Jaffe. Algorithms for Finding Paths with Multiple Constraints. *Network Journal*, 14:95–116, 1984.

[10] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, i P. Tran-Gia. SDN-Based Application-Aware Networking on the Example of YouTube Video Streaming. *2nd European Workshop on SoftwareDefined Networks*, stranice 87–92, 2013.

[11] T. Korkmaz i M. Krunz. Multi-constrained optimal path selection. *The 20th Annual Joint Conference of the IEEE Computer and Communications Societes*, stranice 834–843, 2001.

[12] T. Korkmaz i M. Krunz. A randomized algorithm for finding a path subject to multiple QoS requirements. *Computer Networks*, 36:251–268, 2001.

[13] P. Van Mieghem i H. De Neve. TAMCRA: A Tunable Accuracy Multiple Constraint Routing Algorithm. *Alcatel Corporate Research*, stranice 667–679, 2000.

[14] P. Van Mieghem, H. De Neve, i F. Kuipers. Hop-by-Hop quality of service routing. *Computer Networks*, stranice 407–423, 2001.

[15] P. Van Mieghem, F.A. Kuipers, T. Korkmaz, M. Krunz, M. Curado, E. Monteiro, X. Masip-Bruin, J. Sole-Pareta, i S. Sanchez-Lopez. Quality of Service Routing. *Quality of Future Internet Services*, 2856(1):80–117, 2003.

[16] D. Romić. Uspostava višemedijske sjednice kroz emuliranu programski upravljanu komunikacijsku mrežu, 2014. FER, Diplomski rad.

[17] M. H. Vejlo i N. F. Andersen. Flow-based Dynamic Queue Selector, 2014. Aalborg University, Department of Electronic Systems.

[18] X. Yuan. Heuristic algorithms for multiconstrained quality-of-service routing. *IEEE/ACM Transactions of Networking*, 10(2):244–256, 2002.

**Usmjeravanje u programski upravljanoj mreži zasnovano na parametrima kvalitete višemedijske usluge**

## Sažetak

U programski upravljanim komunikacijskim mrežama (engl. *Software Defined Networks, SDN*), funkcija prosljeđivanja prometa u mrežnim uređajima odvaja se od funkcije upravljanja prosljeđivanjem koja se izvodi na upravljačkom uređaju (engl. *Controller*).

U sklopu upravljačkog uređaja *OpenDaylight* načinjeno je proširenje *QoSRouting* koje koristeći heuristički pristup optimizacije kolonijom mrava radi usmjeravanje zasnovano na kvaliteti usluge. Dobiveni rezultati usporedbe s usmjeravanjem Dijkstrinim algoritmom pokazuju da predloženi algoritam ostvaruje bolju kvalitetu usluge.

**Ključne riječi:** komunikacijske mreže, programski upravljane komunikacijske mreže, usmjeravanje, kvaliteta usluge, optimizacija kolonijom mrava

**Routing based on multimedia service quality parameters in software-defined network**

## Abstract

In Software Defined Networks (SDN) the data plane is separated from the control plane.

With OpenDaylight, an SDN controller, an extension was made that uses a heuristic approach of ant colony optimization to determine routes based on quality of service. The obtained results of comparison to the classical Dijkstra's algorithm show that the proposed algorithm achieves better quality of service.

**Keywords:** networking, software defined networks, routing, quality of service, ant colony optimization