UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET

Slavica Tomović

ALGORITMI RUTIRANJA ZA PODRŠKU KVALITETA SERVISA U SOFTVERSKI DEFINISANIM MREŽAMA

MAGISTARSKI RAD

Podgorica, 2015.

PODACI I INFORMACIJE O MAGISTRANTU

Ime i prezime: **Slavica Tomović**

Datum i mjesto rođenja: 05.02.1991., Nikšić

Prethodno završene studije:

Elektrotehnički fakultet, osnovne akademske studije (180 ECTS kredita), studijski program: Elektronika, telekomunikacije i računari, 2012.

Elektrotehnički fakultet, specijalističke akademske studije (60 ECTS kredita), studijski program: Elektronika, telekomunikacije i računari, smjer Telekomunikacije, 2013.

INFORMACIJE O MAGISTARSKOM RADU

Elektrotehnički fakultet

Studijski program: Elektronika telekomunikacije i računari, smjer Telekomunikacije

Naslov rada: ALGORITMI RUTIRANJA ZA PODRŠKU KVALITETA SERVISA U SOFTVERSKI DEFINISANIM MREŽAMA

Mentor: Prof. dr Igor Radusinović

UDK, OCJENA I ODBRANA MAGISTARSKOG RADA

Datum prijave magistarskog rada: 26.06.2014.
Datum sjednice Vijeća na kojoj je prihvaćena tema: 28.08.2014.
Komisija za ocjenu teme i podobnosti magistranta:

1. Prof. dr Milica Pejanović-Đurišić
2. Prof. dr Igor Radusinović
3. Doc. dr Milutin Radonjić

Komisija za ocjenu rada:

1. Prof. dr Milica Pejanović-Đurišić
2. Prof. dr Igor Radusinović
3. Doc. dr Milutin Radonjić

Komisija za odbranu rada:

1. Prof. dr Milica Pejanović-Đurišić
2. Prof. dr Igor Radusinović
3. Doc. dr Milutin Radonjić

Datum odbrane: _____
Datum promocije:_____

APSTRAKT

SDN (*Software Defined Networking*) predstavlja novi model mrežne arhitekture kod koje je kontrolna ravan razdvojena od ravni podataka i centralizovana na programabilnom kontroleru. U ovom radu je analiziran potencijal SDN-a za garanciju kvaliteta servisa (QoS - *Quality of Service*) na mrežnom nivou. Analizirani su ključni QoS algoritmi rutiranja do sada predloženi u literaturi, i predloženo novo rješenje koje je prilagođeno karakteristikama SDN mreža. Kao ograničenja pri izboru ruta predstavljeni algoritmi uzimaju u obzir dvije metrike: propusnost i kašnjenje na putanji. Pretpostavka je da informacije o dostupnoj propusnosti kontroler dobija prikupljanjem statističkih podataka iz mreže putem OpenFlow protokola. Pored ovih dinamičkih informacija, prilikom analize pretpostavljeno je da su poznata propagaciona kašnjenja na linkovima i izvori i destinacije QoS saobraćaja. Ove pretpostavke se lako mogu opravdati u praksi, a značajno doprinose poboljšanju mrežnih performansi. Istraživanje je motivisano potrebom za dinamičkim uspostavljenjem virtuelinih tunela u *backbone* mrežama NSP (*Network Service Provider*) operatora, gdje QoS zahtjevi korisnika dolaze jedan po jedan i matrica saobraćaja nije unaprijed poznata. U tom kontekstu, SDN je predložen kao zamjena za današnje MPLS (*Multi Protocol Label Switching*) mreže u kojima za sada ni jedan od velikih NSP providera ne koristi mehanizme za inženjering saobraćaja zbog nestabilnosti i velike kompleksnosti upravljanja. Pored algoritama rutiranja QoS soabraćaja, u radu je predstavljen i novi algoritam za rutiranje *best-effort* saobraćaja u cilju minimizacije negativnog uticaja rezervacije resursa. Značajan doprinos rada je praktična realizacija kontrolnog okruženja koje omogućava primjenu QoS mehanizama u SDN mrežama. Dostupna *open-source* rješenja SDN kontrolera nemaju implementiranu podršku za garanciju kvaliteta servisa već saobraćaj prosleđuju najkraćom putanjom, što ne omogućava efikasno korišćenje mrežnih resursa. Stoga, detaljna analiza performansi različitih QoS algoritama je od naročitog značaja za dalji razvoj ove tehnologije.

ABSTRACT

SDN (*Software Defined Networking*) is a new model of network architecture where the network control plane is decoupled from the data plane and centralized at programmable controller. In this paper potential of SDN for guaranteeing quality of service (QoS) at the netwok layer was investigated. The key QoS routing algorithms proposed in literature were analysed, and new solution customized to SDN environment was proposed. Two routing constraints were considered: throughput and path delay. The SDN controller obtains information about available capacity on the links by gathering statistical information via OpenFlow protocol. Beside this, in the analysis it was assumed that propagation delays are known, as well as position of the ingress/engress nodes in the network. These assumptions are justifiable in practise and lead to significant improvement in network performance. The research is motivated by the need to dynamically set up virtual tunnels in NSP (Network Service Provider) backbone networks, where QoS requests arrive one by one and traffic matrix is not known in advance. In that context, SDN is proposed as replacement for today's MPLS (*Multi Protocol Label Switching*) networks, in which none of the bigger NSP providers uses traffic engineering mechanisms due to instability and high management complexity. In addition to QoS routing algorithms, in the paper a new algorithm for routing best-effort traffic was presented, in order to reduce negative impact of resource reservation on its performance. Important contribution of the thesis is practical realization of the control framework that enables QoS mechanisms to be deployed in SDN networks. Available open-source solutions for SDN controller lack support for quality of service provisioning and use shortest path routing instead, which does not allow efficient use of network resources. Therefore, a comprehensive performance analysis of different QoS routing algorithms is of particular importance for the further evolution of this technology.

.

Sadržaj

1. UVOD

Internet se kao globalna mreža konstantno mijenja. Međutim, modifikacije njegove arhitekture su bile veoma rijetke i može se reći da je najveći dio početnog dizajna ostao nepromijenjen. Postojeća Internet arhitektura je bazirana na diskretnim skupovima protokola dizajniranih sa ciljem da obezbijede pouzdanu komunikaciju između udaljenih krajnjih sistema. Akcenat je stavljen na što bržem procesuiranju saobraćaja, pa se odluke o rutiranju donose samo na osnovu destinacione IP (*Internet Protocol*) adrese i bez pružanja prioriteta bilo kojem tipu servisa. Prosleđivanje saobraćaja se vrši najkraćom putanjom od izvora do destinacije. Pod najkraćom putanjom podrazumijeva se putanja sa najmanjim težinskim faktorom, što nije obavazno putanja sa najmanjim brojem hopova. Međutim, ključna karakteristika ovih algoritama je da se težinski faktori linkovima dodjeljuju statički, tako da do promjene jednom izračunatih ruta dolazi samo u slučaju promjene u mrežnoj topologiji, na primjer, dodavanja ili ispada linka. Rutiranje najkraćom putanjom za posledicu ima neravnomjernu distribuciju saobraćaja, što može dovesti do zagušenja određenih djelova mreže čak i kad ukupno opterećenje nije naročito veliko.

Ovakav mrežni dizajn bio je prihvatljiv u ranim godinama Interneta, kada nije bio ni približno današnjih razmjera i kada se 90% ukupnog saobraćaja odnosilo na *e-mail* i *file transfer* aplikacije [1]. Sa druge strane, napredak na polju širokopojasnih tehnologija doveo je do ekspanzije raznovrsnih *real-time* multimedijalnih aplikacija, kao što su video konferencije i Internet telefonija, kojima su potrebne garancije performansi da bi ispravno funkcionisale. Za procjenu performansi na komunikacionom kanalu koriste se različite metrike i njhov izbor zavisi od karakteristika aplikacije. Na primjer, interaktivni audio zahtijeva malo kašenjenje i varijaciju kašnjenja (*jitter*) da bi podržao prirodnu konverzaciju, ali nema stroge zahtjeve u pogledu propusnosti. Suprotno tome, *video streaming* zahtijeva prenos velike količine saobraćaja u kratkom vremenskom periodu i stabilne mrežne uslove za kontinualnu reprodukciju. Kako početno kašnjenje može varirati od korisnika do korisnika to je potrebno adekvatno ograničiti varijaciju kašnjenja. Interaktivni video je najzahtjevniji, i kao kombinacija prethodna dva primjera traži visoku propusnost, malo kašnjenje i malu varijaciju kašnjenja.

Neprekidni rast potražnje za multimedijalnim aplikacijama podstakao je poslednjih godina veliki broj istraživača i organizacija da se pozabave problematikom vezanom za garanciju kvaliteta servisa. IETF (*Internet Engineering Task Force*) je predložio nekoliko modela i mehanizama među kojima su: IntServ (*Integrated Services* [2]), DiffServ (*Differentiated Services* [3]) i MPLS [4]. Međutim, ni jedno od ovih rješenja nije se pokazalo zaista uspješnim niti je globalno implementirano. IntServ i DiffServ su dizajnirani u skladu sa distribuiranim, *hop-by-hop* modelom rutiranja i kao takvi nemaju uvid u širu sliku svih raspoloživih mrežnih resursa. MPLS predstavlja djelimično rješenje sa svojom podrškom za inženjering saobraćaja, ali se sporo prilagođava promjenama u mreži i ne omogućava rekonfiguraciju u realnom vremenu [5]. Takođe, navedeni QoS mehanizmi odlikuju se velikom kompleksnošću i zahtijevaju česte intervencije administratora čije greške mogu dovesti do prekida servisa. Iz ovih razloga, danas se problem garancije QoS-a dominantno

rješava fizičkom izolacijom mreža ili kupovinom redudantne opreme. Mreže *data*-centara, na primjer, koriste odvojene mreže za povezivanje sistema za skladištenje i sistema za visoko performantnu obradu podataka. U nekim slučajevima pribjegava se stepenu redudanse 6 ili više da bi se izbjeglo narušavanje kvaliteta servisa [6]. Takvi pristupi ne samo da zahtijevaju velika ulaganja u infrastrukturu već i nameću velike operativne troškove, dok se sa druge strane najveći dio vremena mrežni resursi neefikasno koriste.

Osnovni razlog neuspjeha predloženih QoS mehanizama leži u previše složenoj kontrolnoj ravan koja je distribuirana u svim mrežnim uređajima. Kao rezultat napora uloženih u prevazilaženje ovog problema nastala je ideja o softverski definisanim mrežama (SDN) [7]. Kod SDN arhitekture kontrolna ravan i ravan podataka su razdvojene, i kontrolna funkcionalnost je prepuštena logički centralizovanom kontroleru. Migracija kontrolne ravni omogućava apstrakciju mrežne infrastrukture. Na ovaj način administratori mogu uz jednostavni programabilni interfejs pokretati nove aplikacije i servise tretirajući cijelokupnu mrežu kao jedinstven entitet, i pri tom pružajući automatsku izolaciju u skladu sa njihovim zahtjevima i prioritetima. Kontrola se obavlja na nivou toka podataka, što otvara prostor za realizaciju dinamičkog, *multipath* rutiranja, koje značajno može doprinijeti povećanju kapaciteta mreže. Na primjer, Google je objavio da je implementacijom SDN-a u svojim WAN (*Wide Area Network*) mrežama za rutiranje saobraćaja između *data*-centara doveo iskorišćenost mrežnih resursa na nivo od čak 95% [8]. Sa druge strane, kontrola na nivou pojedinačnih tokova može se iskorisiti za garanciju performansi od kraja do kraja mreže kroz adekvatne tehnike rezervacije resursa. Ove mogućnosti su detaljno analizirane u magistarskom radu.

Centralna tema magistarskog rada su algoritmimi rutiranja za podršku kvaliteta servisa u SDN mrežama. Istraživanje je velikim dijelom inspirisano potrebom *backbone* NSP operatora za brzim uspostavljanjem virtuelnih tunela koji mogu ispuniti zahtjeve klijenta u pogledu performansi. Kako garancija kvaliteta servisa ima svoju cijenu - NSP operator mora da dodijeli dio fizičkih reursa (kao što su kapacitet linka, prostor za baferovanje) svakom od klijenata, od interesa je da se mrežni resursi iskoriste na što efikasniji način. Različiti profili klijenata imaju različite zahtjeve. Ukoliko je riječ o operatoru interaktivnih multimedijalnih servisa, kašnjenje u mreži je od prevashodnog značaja jer se značajno odražava na doživljaj krajnjih korisnika. Sa druge strane, kod FTP (*File Transfer Protocol*), regularnog *Web* i *Email* saobraćaja kašnjenje nije od velike važnosti, ali je i dalje potrebno rezervisati određeni propusni opseg u mreži s obzirom da se uspostavljanje tunela ne vrši za individualnu sesiju već za agregirani saobraćaj iz nekog domena. U literaturi je predložen veliki broj algoritama rutiranja koji teže da minimiziraju vjerovatnoću blokiranja QoS zahtjeva. Međutim, dominantno ja analiziran scenario kada se vrši samo garancija propusnosti [9-13]. Sa druge strane, garancija kašnjenja je nedovoljno istražena, uprkos značaju koji ima za veliki broj aplikacija. Iz tog razloga, u ovom magistarskom radu akcenat je stavljen na algoritmima rutiranja koji pružaju oba tipa garancija. Radi jednostavnosti samo propagaciono kašnjenje je uzeto u obzir, što nije velika aproksimacija s obzirom da se pokazalo da je u realnim *backbone* mrežama, usled efekta statističkog multipleksiranja, kašnjenje u redovima čekanja neznatno pri opterećenju manjem od 90% [14]. Kako je riječ o linkovima velikog kapaciteta

kašnjenjenje usled prenosa takođe je zanemarljivo. Treba napomenuti da se algoritmi predstavljeni u ovom radu mogu implementirati i u MPLS-TE/SNMP(*Simple Network Management Protocol* [15]) mrežama u kombinaciji sa ASON(*Automatically Switched Optical Network* [16])/GMPLS(*Generalized MPLS* [17])/TL-1(*Transaction Language* 1 [18]) protokolima. Međutim, ogroman broj različitih protokola, njihova distribuirana priroda i potencijalno opasna interakcija čini ova rešenja previše kompleksnim, tako da danas ni jedan od većih mrežnih operatora nema implementiranu neku sličnu funkcionalnost [19].

Rad je organizovan na sledeći način. U drugom poglavlju dat je pregled QoS arhitektura koje su danas zastupljene na Internetu, sa posebnim naglaskom na njihove nedostatke koje je moguće prevazići SDN pristupom. U okviru trećeg poglavlja detaljno je opisana SDN mrežna arhitektura, protokoli koji omogućavaju njenu realizaciju, razlike i prednosti u odnosu na tradicionalni mrežni dizajn. Različiti algoritmi za garanciju kvaliteta servisa predstavljeni su u četvrtom poglavlju. Identifikovani su nedostaci rešenja iz literature i predloženo novo rešenje prilagođeno SDN okruženju. Postignuta poboljšanja u različitim aspektima performansi potvrđena su nizom simulacija. U petom poglavlju predstavljen je originalni, implementirani dizajn SDN kontrolera sa podrškom za garanciju kvaliteta servisa. Pored QoS algoritma, na kontroleru je implementiran i novi algoritam za rutiranje *best-effort* saobraćaja u cilju što manje degradacije njegovih performansi prilikom rezervacije resursa za prioritetne tokove. Performanse predloženog rešenja provjerene su eksperimentalno i upoređene sa IntServ QoS modelom koji se koristi na Internetu. U poslednjem - šestom poglavlju, date su završne konstatacije i zapažanja sa osvrtom na realizovano istraživanje, njegov značaj, kao i mogućnosti rada na budućim istraživanjima u ovom pravcu.

2. QOS ARHITEKTURE

Termin "kvalitet servisa" u literaturi se upotrebljava u kontekstu performansi koje mrežna infrastruktura garantuje mrežnom saobraćaju. To mogu biti garancije u pogledu dodijeljenog propusnog opsega, ograničenja kašnjenja, varijacija kašnjenja, vjerovatnoće gubitaka paketa ili samo dodjela većeg prioriteta pri prenosu. Različiti tipovi garancija potrebni su za mnoge moderne aplikacije da bi ispravno funkcionisale (Tabela 1). Međutim, današnji Internet nije u stanju da odgovori njihovim potrebama, prije svega zbog samog koncepta prosleđivanja na osnovu destinacione IP adrese koji ne omogućava diferenciranje različitih klasa saobraćaja. Pored mogućnosti klasifikacije saobraćaja, garancija kvaliteta servisa zahtijeva implementaciju QoS mehanizama na nivou mrežnog uređaja i na nivou čitave mreže. U zavisnosti od klase saobraćaja kojoj pripadaju, paketi se smještaju u različite bafere. Na nivou uređaja mora se implementirati odgovarajući algoritam koji vrši opsluživanje bafera u skladu sa zahtjevima različitih saobraćajnih klasa u pogledu propusnosti i/ili kašnjenja. Pod pretpostavkom da mrežni uređaji imaju podršku za ovakve mehanizme, aplikacijama se mogu garantovati i performanse na nivou čitave mreže izvršavanjem QoS algoritama rutiranja i protokola komunikacije.

Tabela 1: QoS zahtjevi različitih aplikacija i servisa [20]

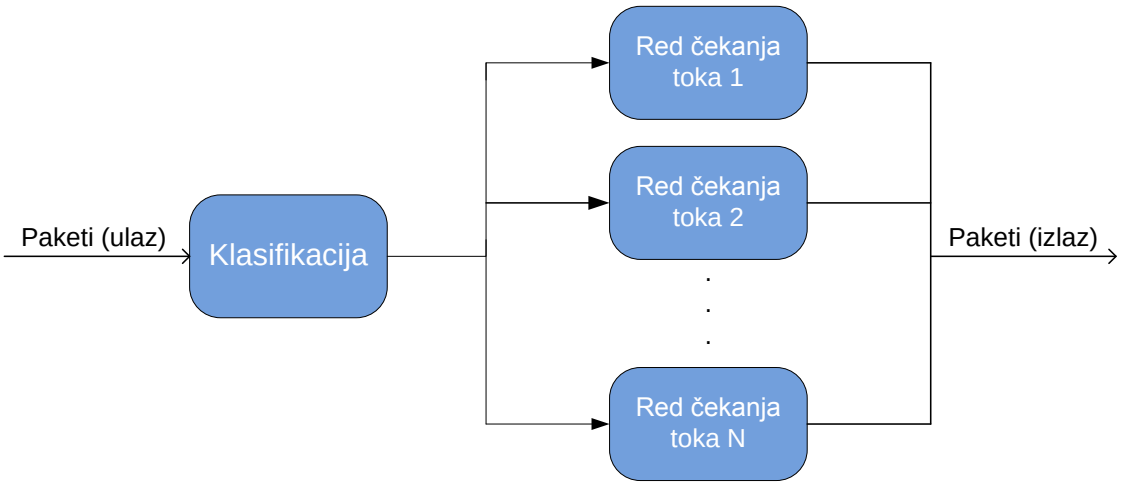
Aplikacija/servis	Brzina prenosa (b/s)	Ograničenje u pogledu kašnjenja	Vjerovatnoća gubitaka paketa
VOIP	64 k	< 150ms	10^{-2}
Video *streaming*	4 - 60 M	< 250ms	10^{-6}
FTP	1 - 10 M	≈10s	0
Video konferencija	128 k - 6 M	< 150ms	10^{-3}
Web	≈ 30 k	2s - 4s	0
Email	≈ 10 k	< 4s	0

Problem garancije kvaliteta telekomunikacionih servisa nije nov i već dugo predstavlja predmet pažnje naučne zajednice. U okviru ovog poglavlja dat je pregled ključnih QoS modela zastupljenih na Internetu. S obzirom da je fokus ovog rada da algoritmima rutiranja, razmatran je samo princip QoS garancija na nivou mreže.

2.1 IntServ

IntServ arhitektura je predložena od strane IETF-a za pružanje apsolutnih QoS garancija na nivou konekcije kroz tehnike rezervacije resursa (propusni opseg, baferi). Kod ovog modela aplikacija eksplicitnom signalizacijom zahtjeva specifičnu vrstu servisa od mreže

prije slanja podataka. Pretpostavlja se da aplikacija šalje podatke tek pošto dobije potvrdu od mreže, i da ti podaci ulaze u okvire opisane u profilu saobraćaja.



Slika 1: Model opsluživanja kod IntServ QoS modela

Trenutno su definisane tri klase servisa u IntServ arhitekturi - najbolji pokušaj (eng. *best effort*), kontrolisano opterećenje (eng. *controlled load*) i garantovani servis (eng. *guaranteed service*). *Best-effort* klasa opslužuje tokove koji ne zahtijevaju QoS od mreže tj. zahtijevaju uslugu koju bi dobili i u slučaju kada mreža ne implementira QoS podršku. Klasa kontrolisano opterećenje ([21]) garantuje tokovima ponašanje nezavisno od trenutnog opterećenja mreže. Drugim riječima, tokovi će primati veoma sličan kvalitet servisa i u slučaju kada je mreža slabo opterećena i kada je mreža zagušena. Stoga, ova klasa je pogodna za aplikacije koje mogu da tolerišu nešto veća kašnjenja, ali su osjetljive na visok nivo zagušenja, tj. zahtijevaju veću pouzdanost prenosa. Aplikacija obavještava mrežu o karakteristikama svog saobraćaja (npr. maksimalnoj veličini *burst*-a i paketa) i potrebnoj propusnosti. Ukoliko se zahtjev prihvati, mreža garantuje specificirane performanse paketima koji se uklapaju u ovaj profil, dok sve ostale ili odbacuje ili tretira kao *best-effort* saobraćaj. Aplikacije koje imaju veoma striktne zahtjeve u pogledu kvaliteta servisa, kao što je VoIP, koriste klasu garantovanog servisa [22]. Ova klasa pruža veoma striktne garancije tokovima, nezavisno od trenutnog stanja mreže, i u pogledu propusnosti i u pogledu kašnjenja. Kao i kod servisa kontrolisanog zagušenja, podrazumijeva se prisustvo mehanizama sa kontrolu pristupa (*Admission Control*) u mrežnim uređajima. Kontrola se može vršiti na nivu sesije ili više agregiranih tokova u zavisnosti od koncentracije saobraćaja u različitim djelovima mreže.

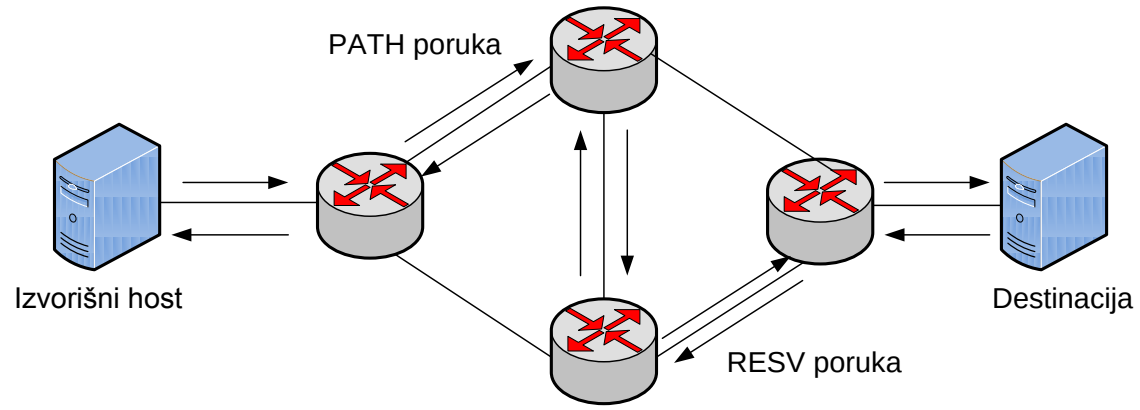
2.1.1 RSVP (*RESOURCE RESERVATION PROTOCOL*) PROTOKOL

IntServ arhitektura podrazumijeva slanje zahtjeva za garanciju kvaliteta servisa, ali ne specificira kako se ovi zahtjevi šalju i obrađuju u mreži. U opštem slučaju moguća su tri rešenja:

- koristiti signalizacini protokol, kao što je RSVP;

- koristiti protokol upravljanja mrežom, kao što je SNMP;
- manuelno konfigurisati uređaje.

Jasno je da je sa aspekta automatizacije ovog procesa prva opcija najprimavljivija. RSVP je *soft-state* protokol za rezervaciju resursa, što znači da će rezervacije u ruterima biti izbrisane ako nisu ažurirane u definisanom intervalu. Originalno je predložen upravo za IntServ mreže, ali je kasnije proširen i za potrebe DiffSev i MPLS QoS arhitektura. Na Slici 2 su prikazane bazične operacije RSVP protkola koje se izvršavaju posredstvom kontrolnih poruka *PATH* i *RESV* [23]. Predajni host šalje *PATH* poruku u kojoj specificira karakteristike IP saobraćajnog toka. Nakon prijema *PATH* poruke, RSVP modul svakog rutera pamti adresu rutera od kojeg je poruka primljena. Prijemni host odgovara *RESV* porukom, koja se prosleđuje istim putem unazad, i rezerviše resurse na svim ruterima duž rute. Konekcija se uspostavlja ukoliko je zahtjev prihvaćen od strane svih rutera na putanji, ili odbija ukoliko bilo koji od rutera nema dovoljno resursa na raspolaganju.



Slika 2: Princip rada RSVP protokola

2.1.2 PREDNOSTI I MANE INTSERV/RSVP MODELA

Značaj IntServ-a je u tome što omogućava apsolutne garancije performansi na nivou pojedinačnih saobraćajnih tokova. Međutim, ovaj pristup ima nekoliko nedostataka. Prvo, mrežni uređaji moraju da održavaju informacije o svakom toku i izvršavaju protokole za signalizaciju, kontrolu pristupa, i kompleksne algoritme raspoređivanja. Samim tim skalabilnost ovog rešenja je kritična. Drugo, zahtjevi za rezervaciju resursa šalju se putanjama koje se određuju nezavisno, standardnim protokolima rutiranja kao što su OSPF (*Open Shortest Path First* [24]) i IS-IS (*Intermediate System to Intermediate System*) [25]. Posledično, može doći do odbijanja QoS zahtjeva ako na izračunatoj putanji nema dovoljno resursa, iako bi on možda mogao biti zadovoljen preko neke druge putanje do iste destinacije. Dalje, bitno je naglasiti da InteServ pruža garancije samo u pogledu maksimalnog kašnjenja u redovima čekanja [26]. Međutim, ukupno kašnjenje u mreži čine još tri komponente:

1. propagaciono kašnjenje - vrijeme koje je potrebno da se signal prenese kroz komunikacioni medijum od izvora do destinacije. Zavisi do dužine linka i karakteristika medijuma za prenos.
2. kašnjenje usled prenosa - vrijeme potebno za smijštanje čitavog paketa na komunikacioni medijum. Zavisi od veličine paketa i kapaciteta linka.
3. kašenje usled obrade u čvorištu - vrijeme koje uređaj potroši na obradu paketa.

Stoga, u cilju postizanja željenih performansi, IntServ korisinici moraju identifikovati adekvatna ograničenja u pogledu dužine bafera na osnovu procjene različitih komponenti kašnjena u mreži.

2.2 DIFFSERV

Kod DiffServ arhitekture [3] umjesto rezervacije resursa na nivou toka podataka, paketi se označavaju na ulazu u mrežu i klasifikuju u konačan broj saobraćajnih klasa. Identifikovanje različitih klasa agregiranog saobraćaja vrši se na osnovu DSCP (*DiffServ Code Point*) polja u IP zaglavlju. Suprotno IntServ modelu gdje se prvo uspostavlja konekcija od kraja do kraja, ovdje ruteri nezavisno sprovode QoS servise na nivou pojedinačnih IP paketa tako što se vrijednost DSCP polja mapira sa odgovarajućim redom čekanja, brzinom njegovog opsluživanja i vjerovatnoćom odbacivanja paketa u slučaju zagušenja.

IETF je do sada standardizovao četiri različite klase servisa (PHBs - *Per Hop Behaviours*):

a) *best-effort* - klasa koja ne podrazumijeva specijalne garancije;
b) EF (*Expedieted Forwarding*) klasa - koristi se za saobraćaj koji je veoma osjetljiv na kašnjenja i koji ne toleriše varijacije kvaliteta servisa (npr. telefonski saobraćaj). Ova klasa ima najveći prioritet pri opsluživanju. Međutim, ako agregirani EF saobraćaj postane prevelik može doći do zapostavljanja ostalog saobraćaja, pa se obavezno na osnovu trenutnih EF tokova u mreži vrši procjena da li treba prihvatiti pakete ovog tipa;
c) AF (*Assured Forwarding*) klasa - koja obezbjeđuje pouzdan i redosledan prenos. Ova klasa dobija bolji tretman od *best-effort* klase, ali kvalitet servisa zavisi od trenutnog stanja u mreži. AF klasa je prikladna za saobraćaj koji zahtijeva potrebnu propusnost ali nije previše osjetljiv na kašnjenje;
d) CS (*Class Selector*) klasa - uvedena je radi kompatibilnosti sa starijim sistemima koji su koristili najviša tri bita u ToS (*Type of Service*) polju IPv4 zaglavlja za označavanje prioriteta saobraćaja. S tim u vezi, definisano je 7 potklasa koje označavaju različite prioritete.

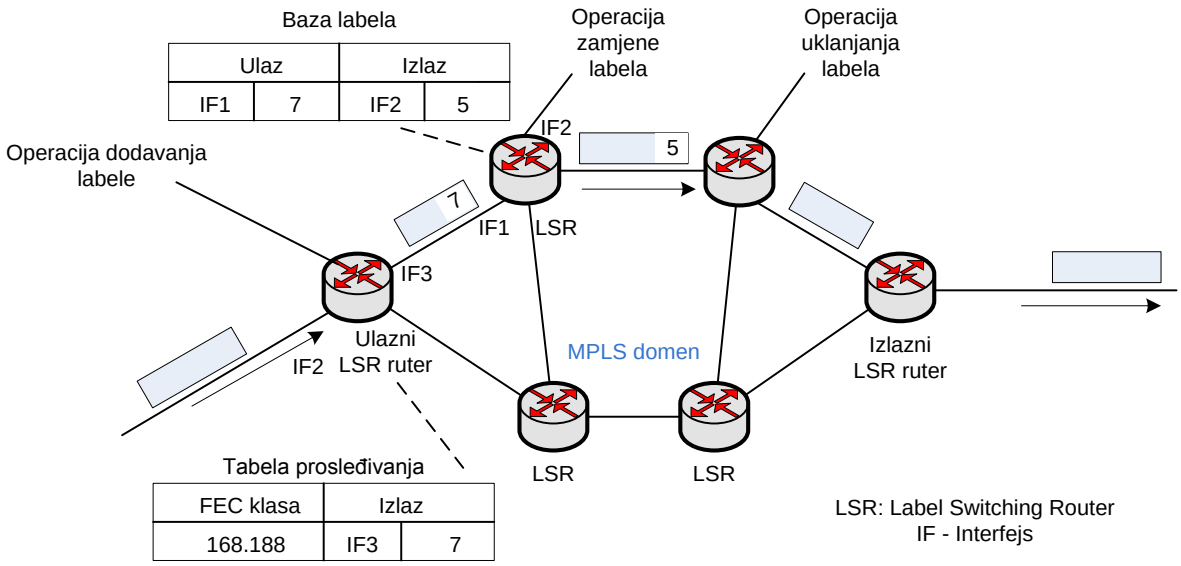
2.2.1 PREDNOSTI I MANE DIFFSERV MODELA

Glavna prednost DiffServ-a je njegova skalabilnost, s obzirom da ruteri moraju da održavaju informacije samo o konačnom broju klasa umjesto o pojedinačnim tokovima. Inteligencija je skoncetrisana na perifernim ruterima, koji pored klasifikacije obavljaju

kontrolu pristupa i oblikovanje saobraćaja u skladu sa administrativnom politikom. Samim tim ovo rešenje je značajno jednostavnije i jeftinije za implementaciju. Međutim, iako se agregacijom tokova u klase eliminiše problem skalabilnosti, na ovaj način se mogu ostvariti samo relativne QoS garancije. Takođe, slično IntServ modelu, kvalitet servisa ograničen je dostupnim resursima na putanjama koje se određuju standardnim metodama rutiranja.

2.3 MPLS

Kao što se iz samog imena može zaključiti, MPLS je tehnologija labeliranja saobraćaja dizajnirana da omogući lakše prihvatanje različitih protokola. Za razliku od tradicionalnog modela usmjeravanja gdje se zaglavlje paketa analizira na svakom hopu do destinacije, MPLS koristi koncept labela za transport paketa kroz mrežu. Labele su zapravo kratke identifikacione oznake paketa fiksne dužine (32 bita). One označavaju kako same putanje kroz mrežu tako i servisne atribute. Kao što je prikazano na Slici 3, informacije iz zaglavlja paketa analiziraju se samo jednom - na ulazu u mrežu. Na osnovu njih se donosi odluka o rutiranju, a dalje se postupak usmjeravanja paketa zasniva samo na labelama koje se dodaju između zaglavlja drugog i trećeg sloja OSI (*Open Systems Interconnection Basic Reference Model*) modela. Labele se konačno uklanjaju iz paketa od strane ivičnog rutera na izlazu iz MPLS domena. Iz datog primjera treba primijetiti da labele imaju lokalni značaj.



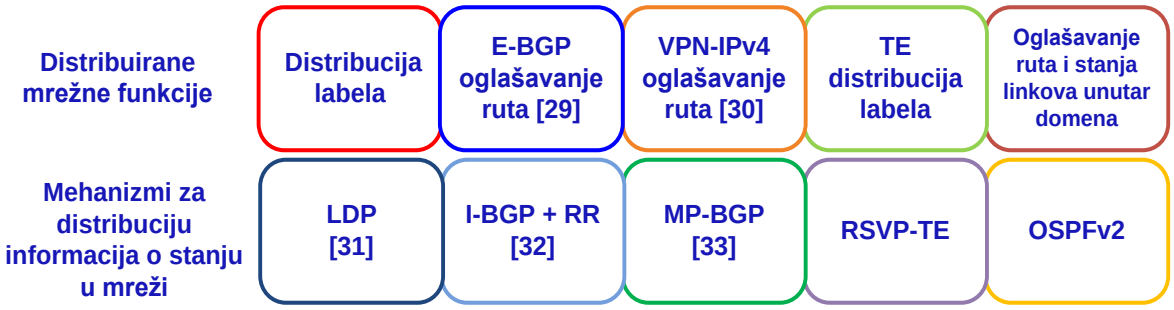
Slika 3: Princip funkcionisanja MPLS mreže

2.3.1 PREDNOSTI I MANE MPLS ARHITEKTURE

Bitna prednost MPLS-a je mogućnost dodjeljivanja labela sa proizvoljnim značenjem. To omogućava korišćenje različitih tehnika prenosa i uspostavljanje odvojenih ruta za različite usluge. S obzirom da je postupak rutiranja nezavisan od mehanizama za prosleđivanje u mreži, QoS garancije i inženjering saobraćaja mogu se vršiti paralelno. Ova mogućnost eksplicitnog uspostavljanja ruta teoretski rešava mnoge nedostatke tradicionalnih

protokola rutiranja koji su ograničeni na analizu samo destinacione IP adrese paketa i ne uzimaju u obzir karakteristike saobraćaja. U tradicionalnim mrežama rutiranje se vrši najkraćom putanjom, pri čemu je metrika rutiranja statička. Osnovni problem takve arhitetkture je što u slučaju zagušenja najkraće putanje između izvora i destinacije često postoje alternativne putanje koje ostaju neiskorišćene. Čak i kada je riječ o *best-effort* modelu servisa, to znači da se mrežni resursi ne koriste efikasno i da postoji potencijal za postizanjem boljeg kvaliteta servisa na istoj mrežnoj infrastrukturi. Kao i u slučaju IntServ-a i DiffServ-a, ukoliko se MPLS oslanja na standardne protokole rutiranja na Internetu, QoS zahtjev (npr. zahtjev servis provajdera za uspostavljanjem virtuelnog tunela) biće odbijen ukoliko na najkraćoj putanji nema dovoljno resursa. Međutim, MPLS je dizajniran da radi sa bilo kojim protokolom mrežnog nivoa, pa se uz sofisticiraniji algoritam rutiranja ovaj problem potencijalno može riješiti. S tim u vezi predložena je ekstenzija RSVP protkola koja omogućava rezervaciju resursa duž eksplicitno navedene rute [27].

Iako naizgled MPLS rješava većinu problema sa kojima se suočavaju tadicionalne IP mreže, mehanizmi za dinamički inženjering saobraćaja nisu implementirani u mrežama današnjih servis operatora [19]. Glavnu prepreku realizaciji ovog koncepta predstavljaju distribuirani protokoli rutiranja, koji imaju problema sa konvergencijom i nestabilnošću u situacijama kada se stanje u mreži mijenja previše brzo i često. Za inteligentno upravljanje saobraćajem potrebne su informacije o trenutnom stanju mrežnih resursa, a u praksi se pokazalo da se distribuiranim protokolima one ne mogu prikupiti ni sa približno dovoljnim nivoom tačnosti. Sa druge strane, MPLS se oslanja na ogroman broj drugih protokola (Slika 4), što upravljanje čini jako kompleksnim i samim tim ne omogućava rekonfiguraciju u realnom vremenu [28].



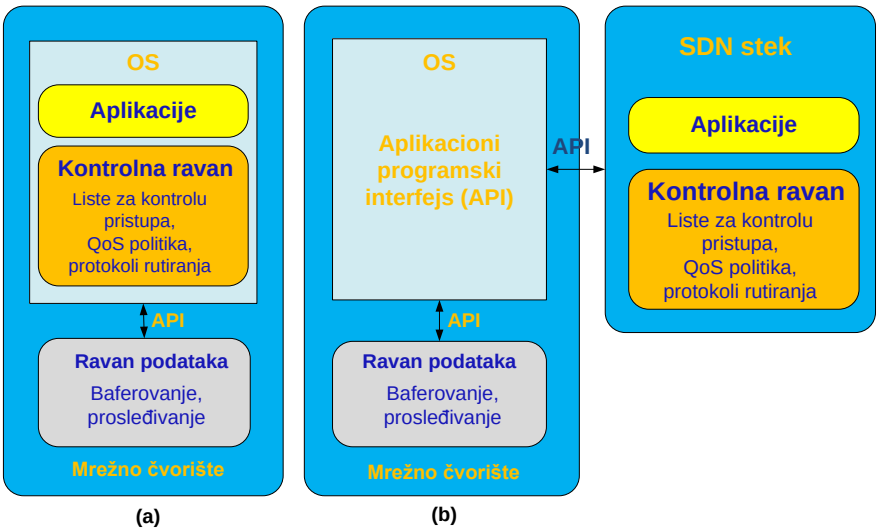
Slika 4: Primjer kontrolne ravni MPLS mreže

3. SOFTVERSKI DEFINISANE MREŽE

U ovom poglavlju objašnjeni su principi softverski definisanog umrežavanja (SDN). Prvo su opisani ključni problemi današnjih mreža koji su bili motivacija za razvoj SDN arhitekture. Zatim je predstavljen model SDN mreže, uključujući ključne elemente i protokole.

3.1 PROBLEMI TRADICIONALNE MREŽNE ARHITEKTURE

Funkcionalnosti mreže organizovane su u ravni podataka, kontrole i upravljanja. Ravan podataka se odnosi na hardver uređaja koji je zadužen za efikasno prosleđivanje saobraćaja. Kontrolnu ravan čine protokoli koji vrše popunjavanje tabela prosleđivanja elemenata u ravni podataka. Ravan upravljanja uključuje sofverske servise, kao što su SNMP-bazirani alati, koji se koriste za udaljeni monitoring i konfigurisanje kontronih funkcija [34]. Mrežna politika definiše se u ravni upravljanja, kontrolna ravna sprovodi tu politiku, a ravan podataka je izvršava prosleđujući saobraćaj na odgovarajući način.



Slika 5: Arhitektura mrežnog uređaja u tradicionalnoj mreži (a) i SDN mreži (b)

U tradicionalnim IP mrežama, kontrolna ravan i ravan podataka su objedinjene, ugrađene u istim mrežnim uređajima, i cijela struktura je veoma decentralizovana (Slika 5a). Takvo rešenje se smatralo važnim za početni dizajn Interneta, jer se činilo kao najbolji način za garantovanje pouzdanosti u mreži, koja je bila ključni cilj dizajna. Štaviše, ovaj pristup se pokazao prilično efikasnim kada su performanse mreže u pitanju, i uspio je da podrži rapidan rast u brzinama prenosa. Međutim, krajnji rezultat danas je veoma kompleksna i relativno statična arhitektura sa aspekta upravljanja ([35-39]), koja nije u stanju da se izbori sa aktuelnim trendovima širenja virtuelizacije, *cloud* i multimedijalnih servisa. Upravljanje se obavlja na vrlo niskom nivou, koji zahtijeva dobro poznavanje same mrežne opreme. Tako, na primjer, svako dodavanje ili uklanjanje uređaja iz mreže zahtijeva od administratora da pristupi pojedinačno velikom broju *switch*-eva, *router*-a, *firewall*-a, *web* portala za

autentifikaciju, kao i da ažurira fajlove za kontrolu pistupa, VLAN (*Virtual Local Area Network*) opcije, QoS politiku i ostale mehanizme zasnovane na protokolima. Dodatno, prilikom rekonfiguracije potrebno je uzeti u obzir mrežnu topologiju, model samih uređaja i verziju njihovog softvera. Za komplikovanije zadatke potrebno je i veće znanje o mreži. Pored kompleksnosti konfiguracije, mrežna okruženja moraju da trpe dinamiku kvarova i prilagode se promjenama saobraćajnog opterećenja. Može se reći da mehanizmi za automatsku rekonfiguraciju gotovo ne postoje u današnjim IP mrežama ([34]), što ne dozvoljava adaptaciju upravljačkih politika u dinamičkim okruženjima. Posledično, greške usled manuelne konfiguracije su veoma česte. Jedan pogrešno podešen uređaj može da uzrokuje neželjeno ponašanje čitave mreže, uključujući između ostalog gubitke paketa, petlje rutiranja, uspostavljanje neželjenih putanja ili kršenje ugovora servisa. Zapravo, iako dosta rijetko, jedan pogrešno konfigurisan ruter može da ugrozi rad čitavog Interneta [40], [41].

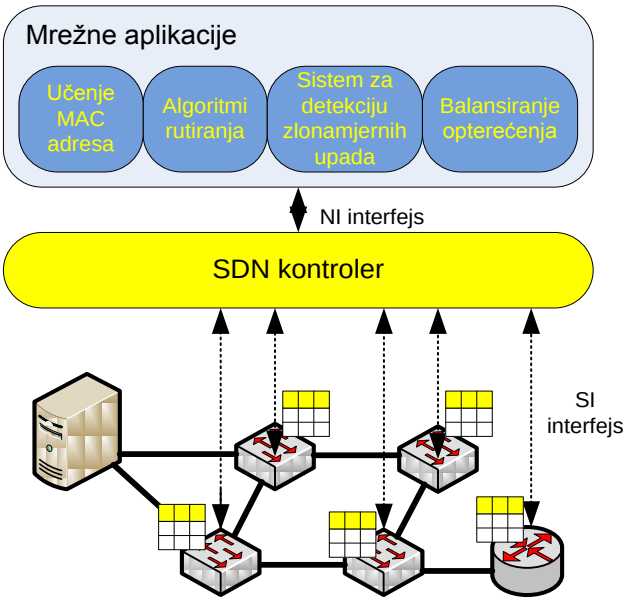
Velike kompanije kao što su Google, Yahoo i Facebook suočavaju se još većim izazovima kada je skaliranje mreže u pitanju. Ovi provajderi servisa koriste masivne algoritme za paralelnu obradu podataka koji su distribuirani preko većeg broja servera koji međusobno razmjenjuju informacije. Kako se djelokrug aplikacija krajnjih korisnika povećava (na primjer, pretraživanje i indeksiranje čitavog *web*-a i vraćanje rezultata pretrage nazad), broj računarskih elemenata koji učestvuje u obradi zahtjeva naglo raste, a podaci koje tom prilikom međusobno razmijene mogu se mjeriti u petabajtima (10^{15} bajtova). Ovim kompanijama potrebne su tzv. hiperskalabilne mreže koje mogu podržati visoko-performantno i ekonomično povezivanje stotina hiljada, potencijalno miliona, fizičkih servera, a taj nivo skalabilnosti nije moguć uz manuelnu konfiguraciju.

Situaciju u današnjim mrežama dodatno komplikuje vertikalna integrisanost kontrolne ravni i ravni podataka, koja ometa inovacije i evoluciju mrežene arhitekture. O tome najbolje svjedoči činjenica da je prelazak sa IPv4 na IPv6 započeo prije više od decenije a još uvijek je u velikoj mjeri nepotpun. Kao što je diskutovano u prethodnom poglavlju, potreba za izmjenom postojećih protokola rutiranja i mrežnog dizajna uveliko je prepoznata. Međutim, zbog inertnosti današnjih IP mreža, za dizajniranje, testiranje i implementaciju novog protokola rutiranja potrebno je između 5 i 10 godina [34]. Slično, *clean-state* promjena mrežne arhitekture predstavlja težan zadatak - jednostavno neizvodljiv u praksi [34][36][37]. Na kraju, ovakva situacija se odrazila na kapitalne i operativne troškove održavanja IP mreže. Da bi se ublažio nedostatak ugrađenih funkcionalnosti, u današnjim mrežama koristi se ogroman broj specijalizovanih komponenti i tzv. *middlebox* uređaja, kao što su *firewall*-i, sistemi za detekciju zlonamjerih upada i sistemi za dubinsku inspekciju paketa. Nedavno istraživanje na nivou 57 poslovnih mreža, pokazalo je da je broj ovih uređaja već u rangu broja rutera u mreži [34][42]. Iako njihovo prisustvo pomaže efikasnijem radu mreže, sa druge strane značajno komplikuje dizajn i upravljanje istom.

3.2 SDN ARHITEKTURA

Logička šema SDN arhitekture prikazana je na Slici 6. Fundamentalna ideja ovog rešenja je razdvajanje kontrolnih funkcija od fizičkih uređaja koje kontrolišu. Mrežna inteligencija je

centralizovana na programabilnom SDN kontroleru koji se kao softverska platforma može izvršavati na serveru opšte namjene. Sama mreža je programabilna, i njeno ponašanje se kontroliše aplikacijama koje se izvršavaju na kontroleru. Ovo je osnovna odrednica SDN-a, koja se smatra njegovom glavnom prednošću.



Slika 6: Logička šema SDN mrežne arhitekture

Važno je naglasiti da SDN arhitektura podrazumijeva samo logičku centralizaciju mrežne kontrole. Kontrolne funkcije se mogu izvršavati na više fizički razdvojenih kontrolera u cilju garancije odgovarajućeg nivoa performansi, skalabilnosti i pouzdanosti [43][34]. Prednosti logičke centralizacije kontrolnih funkcija su mnogostruke:

- Sprovođenje mrežne politike je znatno jednostavnije i manje sklono greškama nego su slučaju manuelne konfiguracije na nivou uređaja;
- Kontrolni program može automatski da reaguje na promjene u mreži ne zahtijevajući bilo kakvu intervenciju;
- Mrežni uređaji više ne moraju da izvršavaju ogroman broj raznovrsnih protokola već samo prihvataju instrukcije SDN kontrolera;
- SDN kontroler ima uvid u stanje svih mrežnih resursa, što omogućava razvoj sofisticiranijih mrežnih funkcija, servisa i aplikacija.

U poslednje vrijeme SDN se sve više spominje kao bitan element u razvoju bežičnih mreža [44][45], gdje razdvajanje kontrolne ravni od ravni prosleđivanja nudi dodatne mogućnosti.

Kao što je prikazano na Slici 6, SDN kontroler podržava dva različita tipa API (*Application Programming Interface*) interfejsa: SI (*Southbound Interface*), koji definiše protokol za komunikaciju sa uređajima u ravni podataka, i NI (*Northbound Interface*) koji se koristi za razvoj samih aplikacija. Tipično, NI interfejs apstrahuje set instrukcija koje koristi SI interfejs za programiranje mrežnih uređaja. Uz otvoreni NI interfejs, SDN aplikacije mogu

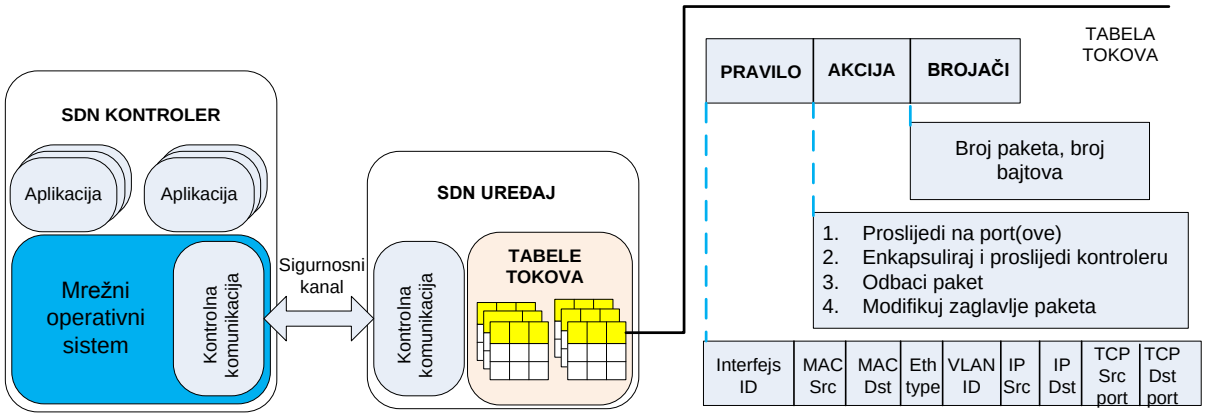
se dizajnirati i pokretati bez uvida u detalje implementacije mrežnih servisa i mogućnosti koje koriste. Kada je SI interfejs u pitanju, nekoliko rešenja je predloženo: OpenFlow [46], ForCES [47], POF [48], OpFlex [49], OpenState [50], ROFL (*Revised OpenFlow Library*) [51], HAL (*Hardware Abstraction Layer*) [52] i PAD (*Programmable Abstraction of Datapath*) [53]. Danas se u komercijalnim rešenjima i za potrebe istraživanja dominantno koristi OpenFlow, pa će u nastavku poglavlja akcenat biti na SDN/OpenFlow mrežama.

3.3 OPENFLOW

OpenFlow je prvi standardizovani protokol koji je omogućio komunikaciju između kontrolne ravni i ravni podataka SDN arhitekture. Omogućava direktan pristup ravni podataka i manipulaciju mrežnim uređajima poput *switch*-eva i rutera, bilo fizičkih bilo virtuelnih. Implementiran je na obije strane interfejsa između mrežne infrastrukture i SDN kontrolnog softvera, i koristi koncept "tokova" (eng. *flows*) za identifikaciju saobraćaja u mreži na osnovu pravila poklapanja koje dinamički ili statički određuje SDN kontroler. Kako je OpenFlow mrežu moguće programirati na nivou "toka" podataka, na ovaj način se ostvaruje visoko-granularna kontrola koja omogućava brzu adaptaciju mreže aplikacijama i korisničkim zahtjevima. Današnje mreže bazirane na IP protokolu ne pružaju ovu mogućnost, jer podrazumijevaju da se svi saobraćajni tokovi između dva krajnja sistema prenose istim putem, bez obzira što su njihovi zahtjevi možda potpuno različiti, kao na primjer u slučaju prenosa video i regularnog *web* sadržaja.

3.3.1 OPENFLOW *SWITCH*

Na Slici 7 prikazana je arhitektura OpenFlow mreže čiji su glavni elementi *switch*-evi koji podržavaju OpenFlow protokol i kontroler u kojem je centralizovana mrežna inteligencija. OpenFlow *switch* predstavlja generalizaciju Ethernet *switch*-a kod kojeg je ravan podataka odvojena od kontrolne ravni i apstrahovana tabelom tokova. Eksterna kontrola se obavlja putem sigurnosnog kanala.



Slika 7: Model OpenFlow mreže i format tabele tokova

OpenFlow koristi tzv. tabele tokova (Slika 7) za reprezentaciju tabela prosleđivanja (tj. ravni podataka) *switch*-eva različitih proizvođača. Svaki od zapisa u tabeli tokova sastoji se od tri dijela:

- Pravila – skupa tzv. *match* polja iz zaglavlja paketa koja definišu saobraćajni tok;
- Akcije – koja određuje postupak obrade paketa;
- Brojača – koji se koriste u statističke svrhe i čuvaju informacije o broju paketa ili bajtova prenesenih na nivou svakog interfejsa i toka, kao i informacije o vremenu proteklom od pristizanja poslednjeg paketa toka.

Polja zaglavlja svakog paketa koji OpenFlow *swich* obrađuje upoređuju se *match* poljima zapisa u tabeli tokova. Ukoliko se ustanovi poklapanje, preduzima se specificirana akcija (na primjer prosleđivanje paketa na određeni port, prosleđivanje paketa kontroleru ili odbacivanje paketa). U suprotnom, paket se prosleđuje kontroleru koji definiše sledeće korake obrade na osnovu implementiranog algoritma.

Iako OpenFlow *switch* omogućava procesuiranje paketa linijskim brzinama, ne pruža kontrolu na nivou paketa. Naime, razdvajanjem kontrolne ravni od ravni podataka izgubljena je mogućnost brzog procesuiranja kontrolnih akcija. Stoga, kontrolne akcije ne preduzimaju se za svaki paket, već se u skladu sa različitim vrijednostima u zaglavlju paketi grupišu u tokove i kontrola se vrši na nivou toka. Kada OpenFlow *switch* primi paket koji ne pripada ni jednom od tokova definisanih u tabeli, prosleđuje ga kontroleru. Kontroler određuje kako taj paket treba biti tretiran i dodaje odgovarajuće instrukcije u tabele tokova OpenFlow *switch*-eva. Svaki sledeći paket istog toka neće se prosleđivati kontroleru već se obrađivati u skladu sa primljenim instrukcijama. Na ovaj način napravljen je kompromis između fleksibilnosti i granularnosti kontrole. Ipak, ovaj kompromis predstavio je novi važan koncept – tok paketa. Za definiciju saobraćajnog toka kontroler može da koristi različita polja zaglavlja koja uključuju parametre drugog, trećeg i četvrtog OSI nivoa. Na Slici 6 prikazan je set polja koju podržava verzija 1.0 OpenFlow protokola. Zajedno, ova polja obezbjeđuju definiciju toka, ali pored toga nude mogućnosti agregacije tokova i separacije različitih tipova saobraćaja. Agregacija se ostvaruje tako što se prilikom kreiranja zapisa u tabeli neka od *match* polja ostave praznim. U tom slučaju, smatra se da bilo koja vrijednost tog polja u zaglavlju paketa zadovoljava uslov poklapanja. Na primjer, tok definisan određenom izvorišnom IP adresom obuhvatiće sav saobraćaj koji potiče od uređaja sa tom IP adresom. Ukoliko se dodatno specificira 80 kao broj TCP porta, ovaj tok obuhvataće HTTP saobraćaj koji generiše taj uređaj. Na isti način i bilo koji drugi tip saobraćaja koji se uvijek očekuje na određenom portu jednostavno se može definisati kao poseban tok.

Kada su akcije koje se sprovode nad paketima u pitanju, OpenFlow trenutno podržava sledeće mogućnosti:

1. Prosleđivanje na određeni port. Na ovaj način se paketi rutiraju kroz mrežu.
2. Enkapsulaciju i prosleđivanje kontroleru. Paketi se isporučuju sigurnosnom kanalu, gdje se vrši njihova enkapsulacija, a zatim se šalju kontroleru kome se prepušta odlučivanje o daljim akcijama.

3. Odbacivanje paketa toka. Može se koristiti u sigurnosne svrhe za sprečavanje DDoS (*Distributed Denial-of-Service*) napada i kontrolisanje *broadcast* saobraćaja.
4. Modifikaciju polja u zaglavlju paketa. Pomoću ove akcije moguće je značajno proširiti mogućnosti *switch*-a. Jedna od mogućih primjena je implementacija NAT (*Network Address Translation*) tabela.
5. Plavljenje, tj. slanje kopije paketa na sve portove osim onog na kojem je paket primljen.
6. Mapiranje paketa u određeni bafer izlaznog porta.

3.3.2 SIGURNOSNI KANAL

Sigurnosni kanal je interfejs putem kojeg OpenFlow *switch* razmjenjuje poruke sa kontrolerom. Ove poruke moraju zadovoljavati format definisan OpenFlow protokolom, i prenose se u binarnom formatu preko TCP konekcije. U okviru OpenFlow protokola definisana su tri tipa kontrolnih poruka: kontroler-*switch* poruke, asinhrone i simetrične poruke. Kontroler-*switch* poruke generiše kontroler i one ne zahtijevaju uvijek odgovor *switch*-a. Koriste se za konfigurisanje *switch*-a, upravljanje tabelom tokova, dobijanje informacija o njenom sadržaju i karakteristikama *switch*-a. Asinhrone poruke *switch* šalje kontroleru da bi ga obavijestio o nekom događaju u mreži. Jedna od takvih situacija je kada *switch* primi paket koji ne pripada ni jednom od tokova definisanih tabelom. Tada se asinhronom porukom specijalog tipa (*PACKET_IN* poruka) kontroleru šalje paket (ili samo jedan njegov dio) na ispitivanje. Slanje asinhronih poruka može biti inicirano i isticanjem roka važenja zapisa u tabli tokova, promjenom stanja nekog od portova ili nekim drugim problemom u mreži. Konačno, simetrične poruke šalju se u oba smjera i koriste se za detekciju problema konekcije između OpenFlow *switch*-a i kontrolera.

3.3.3 KONTROLER

Kontroler je ključni uređaj OpenFlow arhitekture. On sprovodi kontrolnu politiku tako što distribuira odgovarajuće instrukcije mrežnim uređajima. Odgovoran je za donošenje odluka o načinu procesuiranja paketa, i upravlja tabelom tokova tako što dodaje ili briše zapise u njoj preko sigurnosnog kanala. Kontroler u suštini centralizuje mrežnu inteligenciju, dok mreža održava distribuiranu ravan podataka preko OpenFlow *switch*-eva. Stoga, kontroler nudi interfejs koji je neophodan za upravljanje, kontrolisanje i administriranje tabela tokova. Tipično, kontroler je instaliran na serveru priliključenom na mrežu, pri čemu su moguće dvije konfiguracije kontrole: centralizovana i distribuirana. Kod prve jedan kontroler upravlja svim *switch*-evima u mreži, a kod druge dva ili više kontrolera su zaduženi za kontrolu dvije ili više grupa OpenFlow *switch*-eva. Centralizovana konfiguracija je problematična jer se u slučaju otkaza kontrolera sve operacije u mreži prekidaju. Kod distribuirane konfiguracije svaki od kontrolera mora posjedovati "mapu" mreže u realnom vremenu kako ne bi došlo do gubljenja paketa. "Mapa" sadrži informacije o topologiji, lokaciji korisnika, *host*-ova, *firewall*-a i ostalih uređaja i servisa. Štaviše, ona uključuje i sva preslikavanja između imena krajnjih sistema i adresa.

Trenutno je dostupno više različitih *open-source* implementacija kontrolera koji su bazirani na Python, C, C++ ili Java programskom jeziku. Pregled karakteristika nekih od najzastupljenijih dat je u Tabeli 2.

Tabela 2: Poređenje OpenFlow kontrolera [54]

Svojstva	NOX	POX	Floodlight	Trema	OpenDaylight
Open-source	Da	Da	Da	Da	Da
Programski jezik	C++	Python	Java	Ruby i C	Java
Datum objave	2008	2011	2011	2011	2014
Aktivan razvoj	Ne	Da	Da	Da	Da
Modularan dizajn	Umjereno	Umjereno	Odlično	Umjereno	Odlično
Web interfejs	Da	Da	Da	Da	Da
OpenFlow verzija	1.0	1.0	1.0 i 1.3	1.0	1.0 i 1.3
Dokumentacija	Slabo	Slabo	Umjereno	Dobro	Umjereno
Referenca	[55]	[56]	[57]	[58]	[59]

4. QOS ALGORITMI I TEHNIKE ZA INŽENJERING SAOBRAĆAJA U SDN MREŽAMA

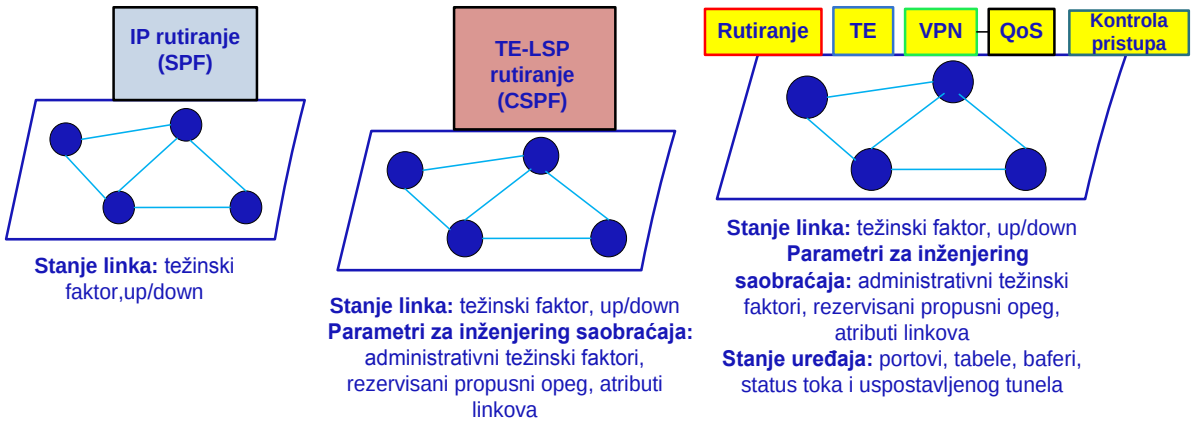
OSPF i ostali dinamički protokoli rutiranja vrše prosleđivanje saobraćaja uvijek najkraćom putanjom. Ovo može biti problematično za tokove kojima su potrebne QoS garancije ukoliko na izračunatoj putanji nema dovoljno resursa. Rešenje predstavljaju QoS algoritmi rutiranja, koji prilikom računanja rute za neki tok uzimaju u obzir njegove zahtjeve u pogledu kvaliteta servisa i uglavnom više različitih parametara mreže, kao što su propunost, propagaciono kašnjenje itd. Kada je mreža slabo opterećena, *best-effort* i QoS servisi se malo razlikuju. U uslovima većeg saobraćajnog opterećenja razlike u performansama postaju značajne. Dok QoS algoritmi pokušavaju da pruže bolji servis tokovima u uslovima zagušenja, tehnike za inženjering saobraćaja raspoređuju saobraćajne tokove tako da se izbjegne zagušenje usled neravnomjerne iskorišćenosti mreže [60]. Dakle, tehnike za inženjering saobraćaja su fokusirane više na poboljšanje performansi i kapaciteta mreže.

U ovom poglavlju analizirani su QoS algoritmi rutiranja i tehnike za inženjering saobraćaja sa aspekta njihove primjenjivosti u SDN mrežama. U prvoj sekciji detaljno je opisan razmatrani model mreže i data su opravdanja za pretpostavke na kojima su bazirani algoritmi predstavljeni u radu. U Sekciji 4.2 predstavljena je terminologija koja se koristi u nastavku rada i osnovni koncepti problema rutiranja. Pregled relevantnih rešenja iz literature dat je u Sekciji 4.3. U okviru iste sekcije predstavljeni su i rezultati poređenja njihovih performansi. U skladu sa identifikovanim nedostacima algoritama, u Sekciji 4.4 predloženo je novo rešenje za inženjering saobraćaja koje se prilagođava karakteristikama QoS zahtjeva. Akcenat je stavljen na postizanju kompromisa između performansi u pogledu kašnjenja i kompeksnosti, s obzirom da je skalabilnost kontrolera jedan od najvećih izazova kada je SDN u pitanju.

4.1 RAZMATRANI MODEL MREŽE

Imajući u vidu probleme postojećih QoS arhitekture na Internetu, koje karakteriše distribuirana kontrolna ravan, u ovom radu istražena je mogućnost pružanja QoS garancija u SDN mrežema. Razmatran je scenario *backbone* mreže OpenFlow *switch*-eva koji su pod administrativnom kontrolom NSP provajdera. Istraživanje je motivisano potrebom servis provajdera za dinamičkim uspostavljanjem saobraćajnih tunela u *backbone* mrežama, pri čemu se pored ispunjavanja njihovih QoS zahtjeva teži maksimizovati kapacitet mreže, što je osnovni interes NSP provajdera.

Do sada je ovaj problem u litraturi razmatran u kontekstu MPLS mreža, koje slično SDN-u vrše kontrolu na nivou saobraćajnog toka. Definicija toka nije toliko fleksibilna kao kod SDN-a, ali omogućava diferenciranje različitih klasa saobaraćaja. Međutim, prilikom poređenja MPLS kontrolne ravni i SDN kontrolne arhitekture važno je napraviti razliku između kocepta mrežne mape, koju koristi prva, i apstrakcije mape koju koristi druga tehnologija.



Slika 8: Mapa mreže u a) IP mrežama b) MPLS-TE mrežama c) SDN mrežama

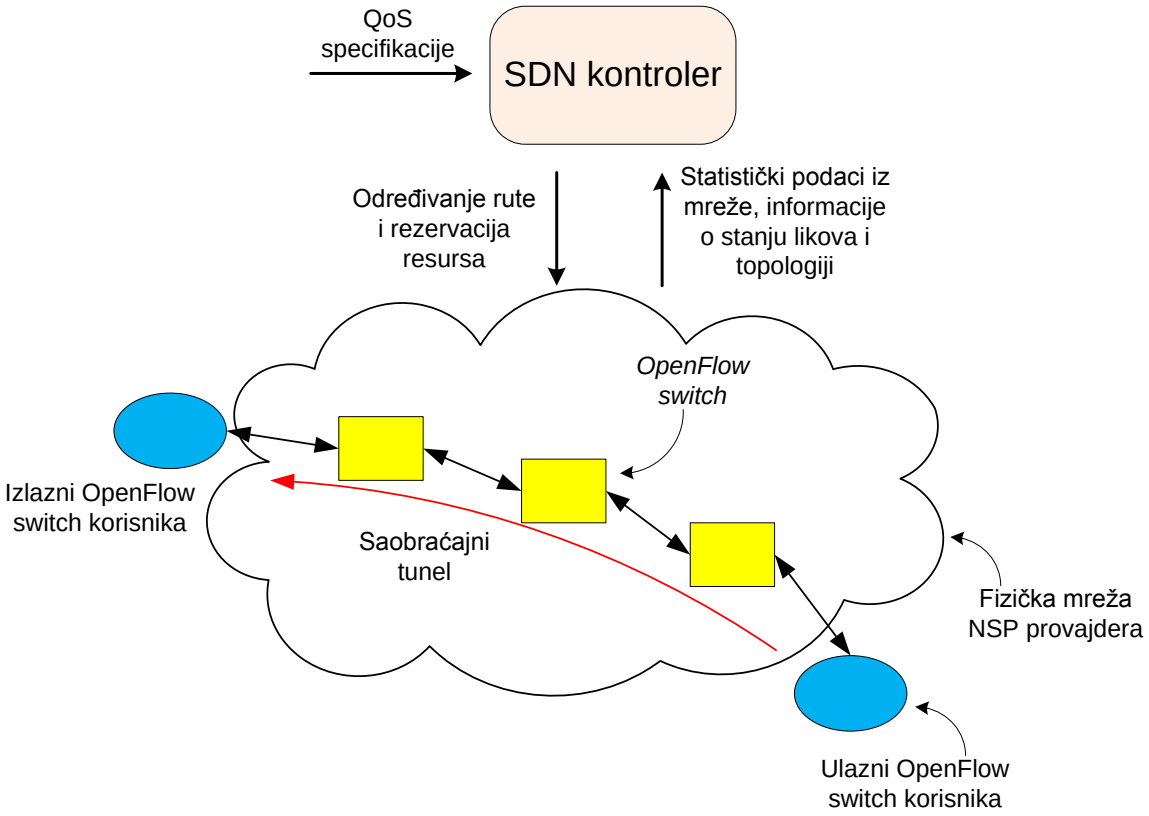
Mapa po definiciji predstavlja graf mrežne topologije. U tradicionalnim IP mrežama protokoli rutiranja kao što su OSPF i IS-IS distribuiraju informacije o stanju linkova do svakog rutera u mreži. Na ovaj način svaki ruter gradi svoju mapu mreže (Slika 8a) koja je konzistentna sa mapama koju održavaju ostali ruteri. Mapa uključuje informacije o rasporedu i povezanosti čvorova, stanju (*up/down*) i težinskim faktorima linkova koji se koriste prilikom računanja ruta. Ove ograničene informacije omogućavaju samo osnovne mrežne funkcije kao što su SPF (*Shortest Path First*) kalkulacije i rerutiranje u slučaju kvara.

U MPLS-TE mrežama koriste se isti distribuirani protokoli rutiranja, ali ovaj put sa ekstenzijom za inženjering saobraćaja koja omogućava prenos većeg broja korisnih informacija o stanju mreže, uključujući: nerezervisani propusni opseg na linku, atribute linka, i administrativne težinske faktore [28]. Mapa se konstruiše na svakom ruteru kao i u tradicionalnim mrežama, ali sada su moguće sofisticiranije kalkulacije ruta koji uzimaju u obzir više ograničenja, ne samo broj hopova.

Kod SDN-a čitava kontrolna ravan je logički centralizovana, pa su dostupne i informacije o tabelama tokova, baferima, portovima, i njihovom stanju i statistikama. Takođe, SDN kontroler posjeduje informacije na globalnom nivou ne samo o stanju linkova, već i o stanju OpenFlow *switch*-eva, ili ako je potrebno stanju tokova paketa ili TE-tunela (Slika 8c). Samim tim, moguća je podrška za inženjering saobraćaja, ali i za ostale mrežne aplikacije kao što su kontrola pristupa, upravljanje mobilnošću, QoS, VPN i mnoge druge. Ukratko, bogate mape i fleksibilna definicija tokova čini SDN mreže više programabilnim. Sa druge strane SDN pruža apstrakciju mape i omogućava kontrolnim programima da se izvršavaju centralizovano, imajući na raspolaganju kompletan pregled mreže, i bez uvida u način na koji se mapa kreira i održava. Dakle, SDN može pružiti sve servise koje MPLS danas pruža, ali što je najvažnije i više od toga. Servisi se mogu globalno optimizovati i učiniti dinamičnijim jednostavnim programiranjem aplikacije na kontroleru. Slojevita protokol arhitetkura je eliminisana, tako da uvođenje novih funkcionalnosti ne zahtijeva bilo kakve promjene u ravni podataka.

Na Slici 9 prikazan je razmatrani model sistema čiju ravan podataka čine OpenFlow

switch-evi. Pretpostavlja se da je kontroleru poznat podskup OpenFlow *switch*-eva koji igraju ulogu ulaznih/izlaznih čvorova, tj. ubacuju/izbacuju saobraćaj iz/u eksterne mreže. Informacije o setu ulaznih/izlaznih čvorova algoritam ruitiranja bi trebao iskoristiti za smanjivanje broja odbijenih zahtjeva usled nedovoljnog kapaciteta mreže. Čak i u situacijama kada se se svaki OpenFlow *switch* ponaša kao ulazni/izlazni čvor, vjerovatno je da je neki podskup čvorova važniji od ostalih, a samim tim algoritam bi trebao da ih zaštiti tako što će njihove QoS zahtjeve odbacivati sa manjom vjerovatnoćom. U zavisnosti od administrativne politike i servisnih ograničenja, može se desiti da je uspostavljanje tunela dozvoljeno samo ka određenim izlaznim čvorovima (npr. VPN saobraćaj). Pretpostavlja se da kontroler posjeduje sve informacije ovog tipa jer korisnici sa njim ugovaraju kvalitet servisa.



Slika 9: Razmatrani model mreže

Kada ulazni čvor primi paket koji ne pripada ni jednom od definisanih tokova, prosleđuje ga kontroleru. Kontroler pretražuje svoju bazu podataka da utvrdi da li je riječ o QoS saobraćaju. Ukoliko jeste, određuje rutu izvršavanjem QoS algoritma i rezerviše resurse duž nje. U svojoj bazi podataka kontroler čuva informacije o svakom uspostavljenjom QoS tunelu, uključujući dodijeljenu rutu i proposuni opseg. Mapa mreže koju kontroler održava pored matrice tokova uključuje informacije o propagacionim kašnjenjima na linkovima i estimiranom nivou zagušenja linkova (i *best-effort* i QoS saobraćajem). QoS zahtjevi dolaze u formatu (*s,t,B,D*), gdje je *s* i *t* označavaju ulazni i izlazni čvor toka redom, a *B* i *D* zahtjeve u pogledu propusnosti i kašnjenja. Pretpostavka je da propagaciono kašnjenje dominantno doprinosi ukupnom kašnjenju na ruti, što se opravdava samim scenariom *backbone* mreže koje zauzimaju ogromna prostranstva. Dalje, kao posledica visokog stepena agregacije

saobraćaja u *backbone* mreži, i velikih brzina prenosa (paketu veličine 1500 B potrebno je 5μs za prenos na 2.5 Gb/s OC-48 linku), očekuje se da će kašnjenja u redovima čekanja biti mala. Istraživanja su pokazala da je usled efekta statističkog multipleksiranja u realnim mrežama ovo kašnjenje neznatno pri opterećenju do 90% [14]. Sa druge strane, algoritmi koji ipak razmatraju kašnjenje u redovima čekanja, najčešće ovu metriku konvertuju u metriku propusnosti, pretpostavljajući da je na uređajima implementiran WFQ (*Weighted Fair Queuing*) ili neki sličan algoritam opsluživanja [61].

4.1.1 ZAHTJEVI ZA SDN ALGORITAM RUTIRANJA

U ovoj sekciji su identifikovani ključni zahtjevi koji SDN algoritam rutiranja mora da zadovolji da bi bio primjenljiv u scenariju koji je prethodno opisan. Na osnovu navedenih kriterijuma izvšen je i izbor algoritama iz literature koji su analizirani u radu.

1. *Online izvršavanje*: Tehnike za inženjering saobraćaja obično pretpostavljaju da je matrica zahtjeva poznata. Međutim, ove tehnike mogu se primijeniti samo prilikom dizajna mreže, ili u mrežama sa stabilnim karakteristikama saobraćaja. Poznavanje saobraćajne matrice omogućava donošenje optimalnijih odluka nego što je to slučaj sa *online* tehnikama, ali na račun veće računske kompleksnosti jer vrše iscrpna pretraživanja prostora rešenja. *Offline* tehnike su fokusirane na što efikasnije korišćenje mrežnih resursa. Jedna od mogućih posledica ovog pristupa je da ponestane dostupnog kapaciteta između određenih ulaznih i izlaznih čvorova, iako to sa nekom drugom šemom rutiranja ne bi bio slučaj. U *offline* modelu rutiranja ovaj problem se neće smatrati bitnim jer se podrazumijeva da su prilikom računanja ruta svi zahtjevi bili poznati i ne očekuju se novi u budućnosti. U scenariju razmatranom u ovom radu, zahtjevi za uspostavljanjem ruta obrađuju se jedan po jedan, po redosledu dolaska. Ne podrazumijevaja se bilo kakvo poznavanje budućih zahtjeva. Za to su potrebni *online* algoritmi koji će računati rute u realnom vremenu. Računanje mora biti jednostavno i brzo, i što je moguće više težiti optimalnom.
2. *Centralizovano izvršavanje* - S obzirom da SDN arhitektura podrazumijeva centralizaciju kontrolne ravni, samo centralizovani algoritmi koji koriste informacije o stanju mreže su analizirani u nastavku.
3. *Mala vjerovatnoća odbacivanja zahtjeva nakon otkaza linka*: U slučaju otkaza linka važno je pronaći alternativne rute za što je moguće više tokova. Ukoliko između određenih ulaznih/izlaznih čvorova više nema dostupnog kapaciteta, onda rerutiranje nekon otkaza linka nije moguće.
4. *Mala računska kompleknost* - Kao što će biti objašnjeno u narednoj sekciji, veliki broj QoS problema rutiranja ne može se riješiti u realnom vremenu (*NP-complete* problemi [62]). S obzirom da SDN kontroler pored zahtjeva za uspostavljanjem QoS-tunela mora da određuje i rute za *best-effort* saobraćaj iz čitave mreže, cilj je pronaći odgovarajući kompromis između složenosti algoritma i njegovih performansi. Iako SDN podrazumijeva samo logičku centralizaciju kontrole, potencijalna interakcija između više kontrolera (od kojih svaki zahtijeva globalni pregled mreže) je vrlo kritična. Iz tog razloga se teži i fizičkoj centralizaciji što većeg stepena.

4.1.2 MODEL GRAFA

U ovoj sekciji predstavljeni su osnovni problemi QoS rutiranja i diskutovana je njihova kompleksnost. Analiza je bazirana je na teoriji grafova. Naime, mreža se može modelovati kao graf *G(N,L)* sa *N* čvorišta i *L* grana koje predstavljaju linkove u mreži. Za link između čvorova *u* i *v* u nastavku rada koristiće se notacija l_{uv}. Svakom linku dodijeljen je težinski faktor $w_i(l_{uv})$ za svaku od metrika rutiranja *i*. Sa $w(P)$ označen je težinski faktor puta $P=(u_1,u_2,u_3...u_l)$.

Metrika definiše tip QoS garancija koji je mreža u stanju da podrži, tako da QoS zahtjev ne može biti podržan ukoliko se ne može mapirati sa kombinacijom izabranih metrika [63]. Metrike koje se uobičajeno koriste za QoS rutiranje podijeljene su u tri kategorije [64]:

1. aditivne - za koje važi da je težinski faktor rute jednak sumi težinskih faktora linkova koji je čine, tj.: $w(P) = w(l_{u_1u_2}) + w(l_{u_2u_3}) + ... + w(l_{u_{l-1}u_l})$
2. multiplikativne - kod kojih je težinski faktor rute jednak proizvodu težinskih faktora linkova, tj.: $w(P) = w(l_{u_1u_2}) \cdot w(l_{u_2u_3}) \cdot ... \cdot w(l_{u_{l-1}u_l})$
3. konkavne - kod kojih važi: $w(P) = \min(w(l_{u_1u_2}), w(l_{u_2u_3}), ..., w(l_{u_{l-1}u_l}))$

Aditivne metrike uključuju kašnjenje, varijaciju kašnjenja, "cijene" linkova i broj hopova. Pouzdanost je multiplikativna metrika, dok je propusnost konkavna. Multiplikativne metrike se mogu svesti na aditivne tako što se težinski fakori linkova w_i i ograničenja *C* zamjene odgovarajućim logaritamski vrijednostima $\log(w_i)$ i $\log(C)$.

4.1.3 PROBLEM RUTIRANJA SA JEDNOM METRIKOM

U najednostavnijem slučaju QoS zahtjevi toka se mogu predstaviti jednom od navedenih metrika. Problem se zatim svodi ili na optimizacioni problem ili na problem ograničenja (eng. *constrained problem*). Same metrike su ili rutom-ograničene ili linkom-ograničene. Konkavne metrike su linkom-ograničene (*eng. link-constrained*) jer je metrika rute određena vrijednošću *bottleneck* linka. Aditivne i multiplikativne metrike su rutom-ograničene (*eng. path-constrained*), jer metrika rute zavisi od svih linkova koji joj pripadaju. U skladu sa poslednjom klasifikacijom, izdvajaju se četiri osnovna problema QoS rutiranja sa jednom metrikom:

1. *problem optimizacije linka*: Za dati graf *G(N,L)* i konkavnu meriku $w(l_{uv})$ za svaki link $l_{uv} \in L$, pronaći put *P* od izvorišnog čvora *s* do destinacionog čvora *t* koji maksimizuje *w(P)*.

Na primjer, pronalaženje rute sa najvećim propusnim opsegom pripada ovaj klasi problema. Za potrebe ovog istraživanja identifikovani problem je riješen modifikacijom Dijkstra algoritma [65], koja će biti objašnjena u narednoj sekciji.

1. *problem ograničenja linka*: Za datu mrežu G(*N*,*L*), konkavnu metriku $w(l_{uv})$ za svaki link $l_{uv} \in L$, i ograničenje *C*, pronaći put *P* od izvornog čvora *s* do destinacionong čvora *t* koji zadovoljava uslov: $w(P)<C$.

Primjer je pronalaženje rute sa zahtijevanim propusnim opsegom. Ovaj problem može se riješiti računanjem optimalnog puta i provjerom ispunjenosti uslova. Drugi pristup je da uklanjanje grana grafa koje ne zadovoljavaju ograničenje, a zatim pokretanje SPF algoritma na novoformiranoj topologiji.

2. *problem otpimizacije puta*: Za dati graf G(*N*,*L*) i aditivnu metriku $w(l_{uv})$ za svaki link $l_{uv} \in L$, pronaći put *P* od izvorišnog čvora *s* do destinacionong čvora *t* koji minimizuje *w(P).*

Ovaj problem se rješava direknom primjenom Dijkstra ili Bellman-Ford algoritma [66]. Pronalaženje rute sa najmanjim brojem hopova, najmanjom cijenom ili kašnjenjem pripada ovoj klasi problema jer su sve pomenute metrike aditivnog karaktera.

2. *problem ograničenja puta*: Za dati graf G(*N*,*L*), aditivnu metriku $w(l_{uv})$ za svaki link $l_{uv} \in L$, i zahtijevano ograničenje *C* pronaći put *P* od izvorišnog čvora *s* do destinacionong čvora *t* koji zadovoljava uslov: $w(P)<C$.

Primjer problema je pronalaženje rute sa kašnjenjem ispod određene vrijednosti, koji se takođe lako rešava Dijkstra ili Bellman-Ford algoritmom.

4.1.4 PROBLEM RUTIRANJA SA VIŠE METRIKA

Svi navedeni problemi su male složenosti i izvršavaju se u polinomijalnom vremenu. Međutim, često je više metrika potrebno za opis zahtijevnog kvaliteta servisa. U zavisnosti od kombinacije metrika koje se koriste u ovakvim situacijama, problem računanja ruta može biti računski toliko kompleksan da je nepraktičan za korišćenje. Bilo koja kombinacija dvije ili više aditivnih ili multiplikativnih metrika stvara tzv. *NP-complete* problem (nerešiv u polinomijalnom vremenu) [67]. Jedine kombinacije koje omogućavaju računanje polinomijalne kompleksnosti su one koje uključuju konkavnu metriku kao što je propusnost i jednu od ostalih metrika, najčešće kašnjenje ili broj hopova.

U cilju prevazilaženja ovog problema korišćeni su različiti pristupi u literaturi. Jedan od njih je definisanje funkcije za generisanje jedne metrike od više parametara. Osnovna ideja je kombinovanje različitih informacija u jednu mjeru kvaliteta i njeno korišćenje prilikom donošenja odluka o rutiranju. Ovaj pristup u najboljem slučaju može poslužiti kao indikator performansi i ne može se iskoristiti za apsolutne garancije. Sa druge strane, propusnost se često koristi kao jedina metrika i u slučaju kada konekcija ima dodatne zahtjeve u pogledu kašnjenja. U slučaju WFQ i WF^2Q agoritama opsluživanja mapiranje ograničenja u pogledu kašnjenja u ograničenje propusnosti je precizno definisano [61]. Međutim, posledica je često

neefikasno korišćenje mrežnih resursa, jer se za male tokove sa striktnim zahtjevima u pogledu kašnjenja rezerviše znatno veći propusni opseg nego što im je zaista potreban.

U Tabeli 3 prikazani su relevantni kompozitni problemi sa aspeka QoS rutiranja. Dvostruki optimizacioni problemi su izostavljeni, jer se smatraju neracionalnim [68]. Naravno, problem dvostruke optimizacije se može aproksimativno riješiti svođenjem kompozitnog problema na dva problema sa jednom metrikom rutiranja, pri čemu se druga optimizacija vrši ukoliko postoji više od jedne optimalne rute po prvom kriterijumu. Ovaj pristup je korišćen za potrebe simulacije WSP (*Widest Shortest Path*) i SWP (*Shortest Widest Path*) algoritama u ovom magistarskom radu. Može se zaključiti da se već u slučaju dvije metrike rutiranja situacija toliko komplikuje da je veliki broj realnih QoS problema nerešiv u realnom vremenu. U cilju njihovog prevazilaženja predložen je veliki broj heurističnih algoritama, čije se aproksimacije uvijek više ili manje odražavaju na tačnost rezultata. Kao što će se u narednom dijelu poglavlja vidjeti, kombinovanje tehnika za inženjering saobraćaja sa algoritmima za garanciju propusnosti i kašnjenja često rezultuje *NP-complete* problemom, što je prilično u literaturi zanemarivano jer nije razmatrana praktična upotrebljivost algoritma. Takvi algoritmi nisu u skladu sa zahtjevima zacrtanim u Sekciji 4.1.1.

Tabela 3: Računska kompleksnost različitih kombinacija metrika rutiranja

	Optimizacija linka	Ograničenje linka	Optimizacija rute	Ograničenje rute
Optimizacija Linka		Polinomijalna	-	Polinomijalna
Ograničenje linka		Polinomijalna	Polinomijalna	Polinomijalna
Optimizacija rute			-	NP-complete
Ograničenje rute				NP-complete

4.2 *STATE-OF-THE-ART* QOS ALGORITMI ZA GARANCIJU PROPUSNOSTI

Principijalni cilj QoS algoritama je pronalaženje rute koja zadovoljava zahtjeve korisnika i efikasno koristi mrežne resurse. U cilju optimizacije mrežnih performansi QoS algoritmi rutiranja su virtuelno bazirani na dvije tehnike. Prva je fokusirana na uštedu resursa i bira rute sa što manjim brojem hopova koje mogu da ispune tražene zahtjeve u pogledu performansi. Druga bira najmanje opterećen put u cilju balansiranja saobraćaja. Kombinovani oblik optimizacije korišćenja mrežnih resusa ne može se lako realizovati jer su pomenuta dva pristupa u konfliktu. Stoga, neophodni su algoritmi koji uzimaju u obzir više aspekata mreže, pored propusnosti i broja hopova. S obzirom da se takvi pristupi odražavaju na računsku kompleknost, u ovom radu je razmatrano više rešenja u cilju identifikacije potrebnog kompromisa između performansi i brzine izvršavanja algoritma.

4.2.1 MHA ALGORITAM

Najčešće korišćen algoritam za rutiranje QoS tokova je *min-hop* algoritam (MHA) [69], koji računa putanju sa najmanjim brojem hopova koja zadovoljava zahtjev u pogledu propusnosti. Algoritam održava informacije o "dostupnom" (nerezervisanom) propusnom opsegu na svakom od linkova, i razmatra samo one linkove koji imaju dovoljno resursa na raspolaganju. MHA algoritam je veoma jednostavan, ali može brzo da kreira usko grlo za buduće zahtjeve, što dovodi do slabe iskorišćenosti mrežnih resursa.

4.2.2 WSP ALGORITAM

U [70] predložena je varijanta MHA algoritma, poznata kao WSP (*Widest Shortest Path*) algoritam, koji iz seta najkraćih ruta koje mogu da ispune QoS zahtjev bira najširu, tj. onu sa najviše slobodnog kapaciteta. Na ovaj način pokušava se napraviti kompromis između dva oprečna zahtjeva: balansiranja opterećenja i potrošnje resursa. Za potrebe pronalaženja svih najkraćih putanja Dijkstra algoritam je nadogradjen sa dodatnom rekurzivnom funkcijom (Algoritam 1).

4.2.3 SWP ALGORITAM

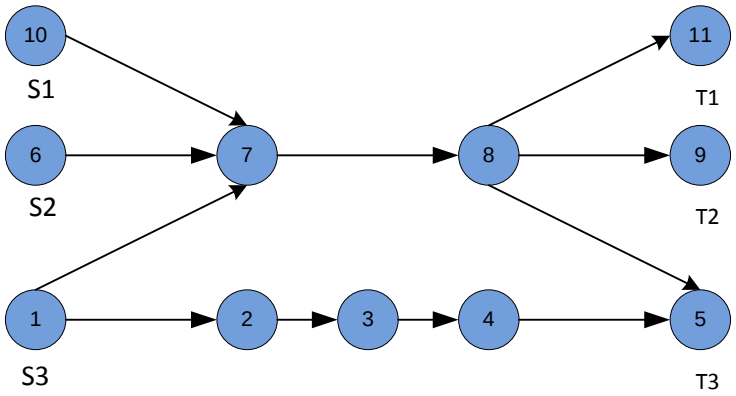
Komplementarna vrsta kompromisa napravljena je sa SWP (*Shortest Widest Path*) algoritmom, koji od svih "mogućih" ruta bira najširu. Ukoliko postoji više takvih ruta, optimalnom se smatra najkraća od njih. Problem pronalažanja "najšire" rute konceptualno je sličan problemu rutiranja najkraćom putanjom. Jedina razlika je što se umjesto rute sa najmanjim brojem hopova bira ona najvećeg kapaciteta. Stoga, logičan korak bi bio primjena nekog od SPF algoritama za ove svrhe. Međutim, primjena rekurzije slične kao kod SWP algoritma može dovesti do stvaranje petlji rutiranja, jer samo *bottleneck* linkovi utiču na izbor putanja [71]. Iz tog razloga, algoritam je implementiran tako da se izvršava u tri faze: prvo se modifikovanim Dijkstra algoritmom određuje kapacitet "najšire" rute, zatim se sa grafa uklanjaju svi linkovi koji nemaju toliko slobodnog kapaciteta na raspolaganju, i na kraju se originalnim Dijkstra algoritmom određuje optimalna ruta.

4.2.4 MIRA ALGORITAM

Prethodno opisani algoritmi koriste informacije o topologiji mreže i slobodnom kapacitetu linkova, ali ne uzimaju u obzir lokaciju ulaznih/izlaznih čvorova. Posledično, izabrana ruta često predstavlja smetnju za buduće zahtjeve, tj. stvara "interferenciju". Na primjer, ako se razmatra topologija sa Slike 10, postoje tri potencijalna para izvor-destinacija: (S1,T1), (S2,T2) i (S3,T3). Pretpostavlja se da svaki link ima 1 jedinicu slobodnog propusnog opsega. U slučaju da stigne QoS zahtjev za rutiranje 1 jedinice saobraćaja između S3 i T3, MHA algoritam odabraće rutu 1-7-8-5 koja blokira parove (S1,T1) i (S2,T2).

Algoritam 1: Računanje svih najkraćih puteva u mreži

```
1:# G - graf sa linkovima koji zadovoljavaju zahtjev u pogledu propusnosti
2:# dist[v] - procijenjena vrijednost težinskog faktora rute do čvora v
3:# preth[v] - prethodni čvor na ruti do čvora v
4:# izv - izvorni čvor, dest - destinacioni čvor, N[G]- čvorovi grafa
 5:
 6:  function inicijalizacija(G,izv):
 7:     sve_najkraće_rute={}
 8:     for v in N[G] do:
 9:        dist[v]=∞
10:        preth[v] = []
11:    dist[izv]=0
12: end function
13:
14: function modifikovana_Dijkstra(G,izv):
15:    čvorovi = N[G]
16:    while čvorovi ≠ {} do:
17:        min_čvor = Null
18:        for u ∈ čvorovi do:
19:           if dist[u] ≠ Null then:
20:              if min_čvor == Null or dist[u] < dist[min_čvor] then:
21:                 min_čvor = u
22:        if min_čvor == Null then:
23:           break
24:        čvorovi = čvorovi - {min_čvor}
25:        trenutni_tf = dist[min_čvor]
26:        for v in min_čvor.susjedi() do:
27:           novi_tf = trenutni_tf + 1
28:           if dist[v] == Null or novi_tf < dist[v] then:
29:              dist[v] = novi_tf
30:              preth[v] = {min_čvor}
31:           else if novi_tf == tf[v] then:
32:              preth[v] = preth[v] + {min_čvor}
33: end function
34:
35: function sviNajkraćiPutevi(G,izv,dest,P):
36:    put = kopija(P)
37:    if (izv == dest) then:
38:        put = put + {izv}
39:        svi_najkraći_putevi = svi_najkraći_putevi + {put.obrunuti_redosled()}
40:    else:
41:        put = put + {dest}
42:    for v ∈ preth[dest] do:
43:         svi_najkraći_putevi(G,izv,v,put)
44: end function
45:
46: function WSP(G,izv,dest):
47:     modifikovana_Dijkstra(G,izv)
48:     sviNajkraćiPutevi(G,izv,dest,{})
49:     return svi_najkraći_putevi
50: end function
```



Slika 10: Ilustrativni primjer [9]

MIRA (*Minimum Interference Routing Algorithm*) [9] je *online* tehnika rutiranja koja teži da minimizuje interferenciju sa potencijalnim budućim QoS zahtjevima između ostalih parova izvor-destinacija. Sam koncept interferencije vezuje se za pojam maksimalnog toka (eng. *maxflow* [72]) iz teorije grafova. Maksimalni tok između izvora *s* i destinacije *t* (Θ_{st}) je broj koji indicira maksimalnu količinu saobraćaja koja se može poslati između ovih čvorova korišćenjem svih mogućih putanja u mreži. Predstavlja gornju granicu za ukupnu brzinu prenosa u bitima po sekundi koja se može u idealnom slučaju ostvariti između *s* i *t*. Ova vrijednost se smanjuje svaki put kada se rutira novi zahtjev između *s* i *t*, ali takođe može da se smanji i kada se rutiraju zahtjevi između ostalih parova ulazno-izlaznih čvorova. Nivo interferencije koji uspostavljanje saobraćajnog tunela unosi za par (s_i,t_i), definiše se kao vrijednost za koju se nivo maksimalnog toka tog para smanji. Cilj MIRA algoritma je pronalaženje rute koja unosi najmanji nivo interferencije, odnosno rute koja maksimizuje sumu maksimalnih tokova ostalih parova izvor-destinacija:

$$\max \sum_{(s,t)\in Parovi/(a,b)} \alpha_{st}\theta_{st} \qquad (1)$$

gdje (a,b) predstavlja par čiji se zahtjev obrađuje. Sa parametrom $a_{s,t}$ reflektuje se značaj para (s,t). Ukoliko je $a_{s,t}$=1 $\forall$ (s,t), onda se svi parovi smatraju ravnopravnim. Kako se ovako formulisan problem optimizacije ne može riješiti u polinomijalnom vremenu, kao aproksimativno rešenje usvojen je pristup rutiranja najkraćom putanjom, pri čemu su težinski faktori linkova definisani na specijalan način:

$$w(l) = \sum_{(s,t)\in Parovi/(a,b)} \alpha_{st} \frac{\partial \theta_{st}}{\partial R(l)} \qquad (2)$$

gdje $\frac{\partial \theta_{st}}{\partial R(l)}$ predstavlja promjenu vrijednosti maksimalnog toka između izvora *s* i destinacije *t* u situaciji kada se slobodni kapacitet linka *l* inkrementalno smanji.

Iz teorije grafova poznato je da je vrijednost maksimalnog toka između dva čvora jednaka kapacitetu tzv. *min-cut* seta [72], koji definiše kritične linkove za ta dva čvora. Ovi linkovi se smatraju kritičnim jer nakon njihovog uklanjanja iz grafa posmatrani čvorovi

ostaju nepovezani. Ukoliko se sa C_{st} označi set kritičnih linkova za par (*s*,*t*), na osnovu *maxflow-mincut* teoreme važi:

$$\frac{\partial \theta_{st}}{\partial R(l)} = \begin{cases} 1, ako\ l \in C_{st} \\ 0\ ako\ l \notin C_{st} \end{cases} \tag{3}$$

pa se problem računanja težinskih faktora može svesti na određivanje kitičnih linkova za svaki ulazno-izlazni par čvorova:

$$w(l) = \sum_{(s,t):\, l \in C_{st}} \alpha_{st} \tag{4}$$

Za određivanje *min-cut* seta može se iskoristiti Goldberg Tarjan algoritam [73]. Na kraju, rutiranje saobraćaja se vrši putanjom sa najmanjim težinskim faktorom.

4.2.5 DORA ALGORITAM

DORA algoritam [10] takođe je baziran na ideji minimizacije interferencije sa budućim zahtjevima. Predložen je u cilju prevazilaženja osnovnih nedostataka MIRA algoritma, a to su: računska kompleksnost i nebalansirano korišćenje mrežnih resursa. Ako se pretpostavi mreža sa *N* čvorova i *L* linkova, u najgorem slučaju kada se svaki čvor ponaša kao izvorni čvor za drugi, MIRA zahtjeva N^2 izračunavanja kritičnih linkova. S obzirom da je kompleksnost algoritma za računanje *min-cut* seta $O(N^3)$ [10], vrijeme izvršavanja MIRA algoritma u najgorem slučaju je $O(N^5)$, što je nekoliko redova veće od kompleksnosti SPF algoritma (npr. *O(LN)* u slučaju Dijkstre). Drugi nedostatak posledica je toga što MIRA u definiciji težinskog faktora ne uzima u obzir saobraćajno opterećenje linkova. Na primjer, u slučaju da između para izvor-destinacija postoji više odvojenih putanja, može se desiti da MIRA rutira zahtjeve stalno istom putanjom, sve dok se njeni resursi potpuno ne istroše.

Rad DORA algoritma podijeljen je na dvije faze: *offline* i *online* fazu. *Offline* faza izvršava se samo pri inicijalizaciji mreže i u slučaju neke promjene u topologiji, i to u sledećim koracima:

1. Za svaki par izvor-destinacija (*s*,*t*) određuje se set nepreklapajućih putanja *NP(s,t).* Jedan od načina je primjena Dijkstra algoritma za pronalaženje najkraće putanje u pogledu broja hopova, i zatim ukanjanje svih njenih linkova iz grafa. Procedura se ponavlja sve dok izvor *s* i destinacija *t* ostanu nepovezani.

2. Za svaki par (*s*,*t*) kreira se niz *OTF(s,t)* dužine *L* čiji se elementi inicijalizuju na 0. Niz *OTF(s,t)* će oslikavati *offline* težinske faktore linkova.

3. Za svaki par *(s,t)* i za svaki link *l*, ispituje se da li link *l* pripada setu nepreklapajućih putanja *NP(s,t).* Ukoliko je to slučaj, vrijednost niza *OTF(s,t)* na poziciji koja odgovara tom linku umanjuje se za 1. Zatim se određuje broj pojavljivanja istog linka u nepreklapajućim putanjama ostalih parova izvor-destinacija. Vrijednost *OTF(s,t)(l)* uvećava se za ovaj broj.

4. Korak 3 ponavlja se za sve ostale parove izvor-destinacija.

5. Vektori *OTF(s,t)* svakog para se normalizuju tako da najmanjem elementu svih *OTF* nizova odgovara vrijednos 0, a najvećem 100.

Definisanjem *OTF* nizova, linkovi se diferenciraju po vjerovatnoći sa kojom će prenositi saobraćaj između nekog ulaznog i izlaznog čvora. Što je vrijednost *OTF(s,t)(l)* veća, to se link *l* smatra kritičnijim i pokušava izbjeći prilikom rutiranja zahtjeva.

Online faza algoritma izvršava se u trenutku pristizanja zahtjeva. Prvo se iz grafa eliminišu linkovi koji ne mogu da zadovolje zahtjev u pogledu propusnosti, a zatim se za ostale linkove računaju težinski faktori prema formuli:

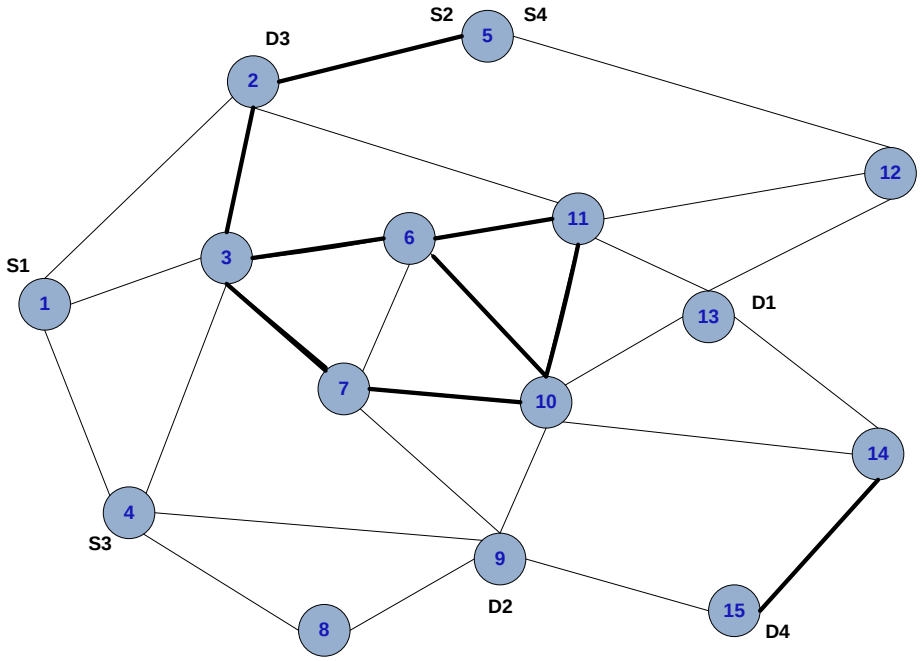
$$w(l) = (1 - BWP) \cdot OTF(l) + BWP \cdot RB^{-1}(l) \tag{5}$$

gdje $RB^{-1}(l)$ predstavlja normalizovanu recipročnu vrijednost slobodnog kapaciteta na linku, a parametar *BWP* reguliše uticaj ovog dinamičkog dijela težinskog faktora ($0 \le BWP \le 1$). Vrijednost *BWP*=0 odgovara slučaju kada se uopšte ne razmatra slobodni kapacitet na linkovima, dok se pri *BWP*=1 ne razmatra njihova kritičnost. Pronalaženje optimalne rute se na ovaj način svodi na pronalaženje najkraće putanje. Dakle, kompleksnost druge faze DORA algoritma odgovara kompleksnosti SPF algoritma. Prva faza je složenija, ali se izvršava samo u slučaju promjene topologije. Maksimalan broj nepreklapajućih putanja između izvora i destinacije je *O(L)*, pa je kompleknost pronalaženja svih nepreklapajućih putanja između *p* parova $O(pNL^2)$. Vrijeme inicijalizacije *OTF* nizova je *O(p)*. Koraci 3 i 4 zahtijevaju provjeru svih linkova posebno za svaki par izvor-destinacija, pa je njihova ukupna kompleksnost *O(pL)*. Konačno za normalizaciju *offline* težinskih fakora potrebno je *O(p)* vremena. U najgorem slučaju kada se svi čvori ponošaju kao i kao izvori i kao destiancije, $p=N^2$ .Stoga, može za zaključiti da je kompleksnost *offline* dijela algoritma dominanto određena prvim korakom koji je tada $O(N^3L^2)$.

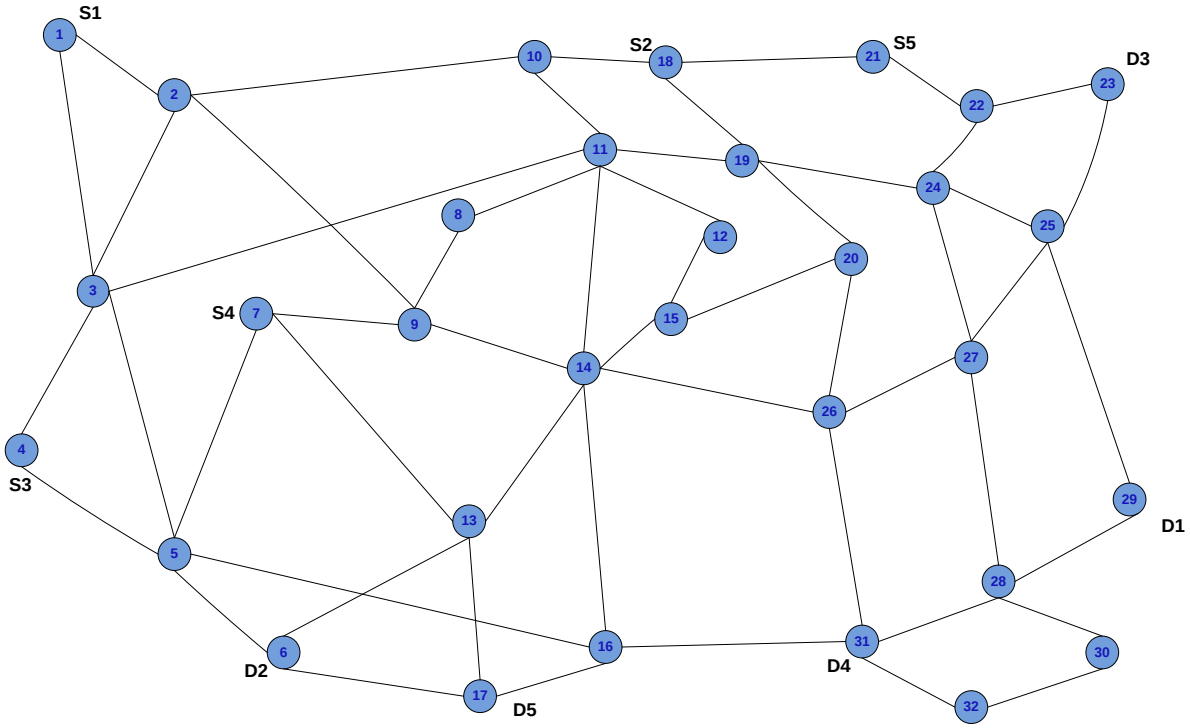
4.3 PERFORMANSE ALGORITAMA ZA GARANCIJU PROPUSNOSTI

U ovoj sekciji predstavljeni su rezultati poređenja prethodno navedenih algoritama rutiranja. Za potrebe analize napravljeno je simulaciono okruženje u Python programskom jeziku. Simulacije su izvšene za dvije topologije. Prva topologija preuzeta je iz radova u kojima su predloženi MIRA i DORA algoritmi [9][10] (MIRA topologija). Sastoji se od 15 čvorova koji su povezani kao što je prikazano na Slici 11. Linkovi označeni podebljanom linijom imaju kapacitet 4800 jedinica, a tankom linijom 1200 jedinica. Vrijednosti su izabrane da modeluju OC-48 i OC-12 linkove. Na topologiji izdvajaju se četiri para izvor-destinacija: (S1,T1), (S2,T2), (S3,T3) i (S4,T4). Druga topologija bazirana je na realnoj ISP(*Internet Service Provider*) topologiji (Verizon mreža na prostoru Sjedinjenih Američkih Država), koja je dopunjena dodatnim linkovima u cilju postizanja većeg nivoa konektivnosti. Sastoji se od 32 čvora od kojih su izvorišni/odredišni odabrani kao što je prikazano na Slici 12.

U eksperimentima generisano je 500 QoS zahtjeva, pri čemu je veličina zahtjeva birana na slučajan način između 10, 20, 30 i 40 jedinica. Simulacije su izvršene više puta u cilju dobijanja vjerodostojnijih rezultata.



Slika 11: MIRA topologija



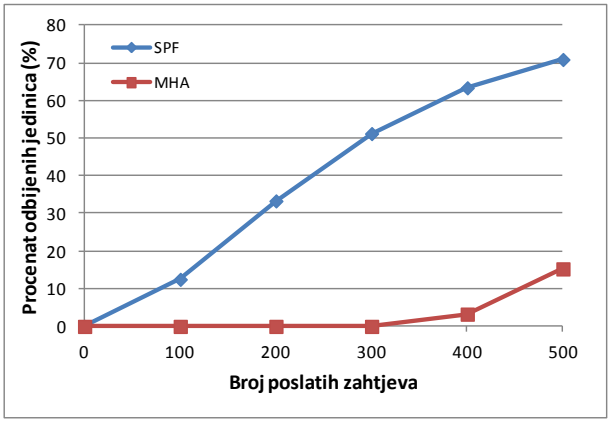
Slika 12: Proširena Verizon topologija

Kao indikator performansi korišćene su dvije metrike: vjerovatnoća blokiranja zahtjeva i srednja dužina rute. S obzirom da se zahtjevi razlikuju po vrijednosti, broj odbijenih zahtijeva ne odražava neophodno dobru efikasnost algoritma. Iz tog razloga, kao prva metrika zapravo razmatrana je vjerovatnoća blokiranja na nivou jedinica propusnog opsega:

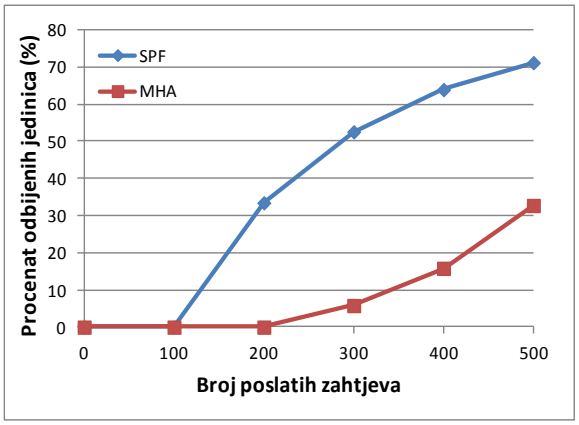
$$vjerovatnoća\_blokiranja\_jediničnog\_zahtjeva = \frac{\sum_{i \in blk} propusni\_opseg(i)}{\sum_{i \in k} propusni\_opseg(i)} \qquad (6)$$

gdje *blk* predstavlja set blokiranih, a *k* set svih primljenih zahtjeva. Kompleksnost je još jedan bitan indikator kvaliteta algoritma, ali ona je već diskutovana u prethodnim sekcijama.

Značaj QoS tehnika za inženjering saobraćaja lako je uočljiv sa Slika 13. i 14., gdje je MHA kao najjednostavniji QoS algoritam upoređen sa SPF algoritmom na koji se oslanjaju današnje QoS arhitekture. Iako oba algoritma koriste broj hopova kao metriku, prvi razmatra informacije o dostupnom kapacitetu na linkovima i ne uzima u razmatranje one koji nisu u stanju da zadovolje zahtjev. Kada je SPF algoritam u pitanju, prvo se računa najkraća putanja (u pogledu broja hopova), i ukoliko ona nema dovoljno resursa na raspolaganju zahtjev se odbacuje. U realnim scenarijima ova procedura provjere se obavlja putem RSVP protokola.



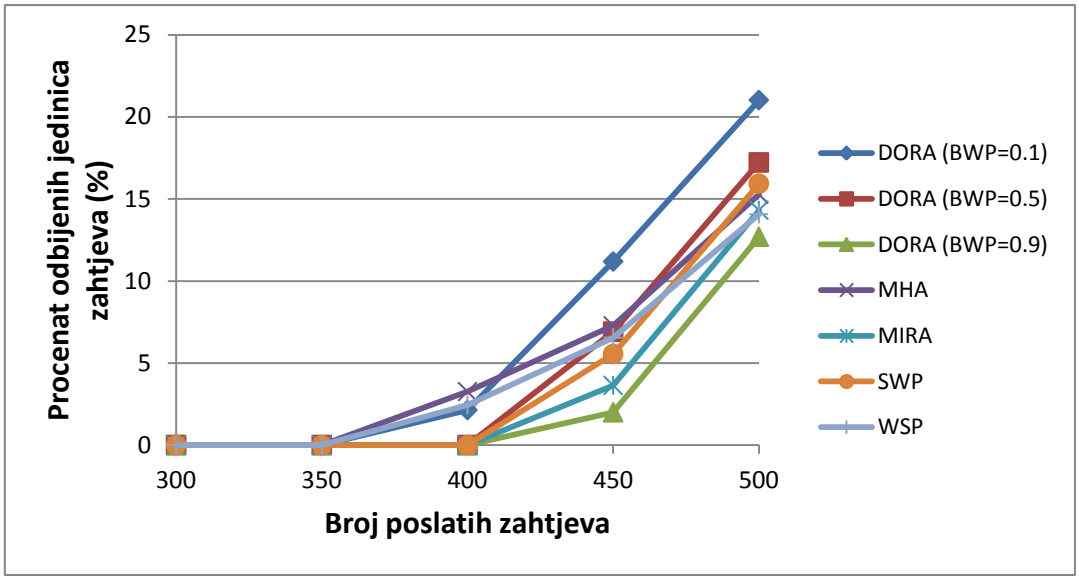
Slika 13: Poređenje MHA i SPF algoritama na MIRA topologiji



Slika 14: Poređenje MHA i SPF algoritama na Verizon topologiji

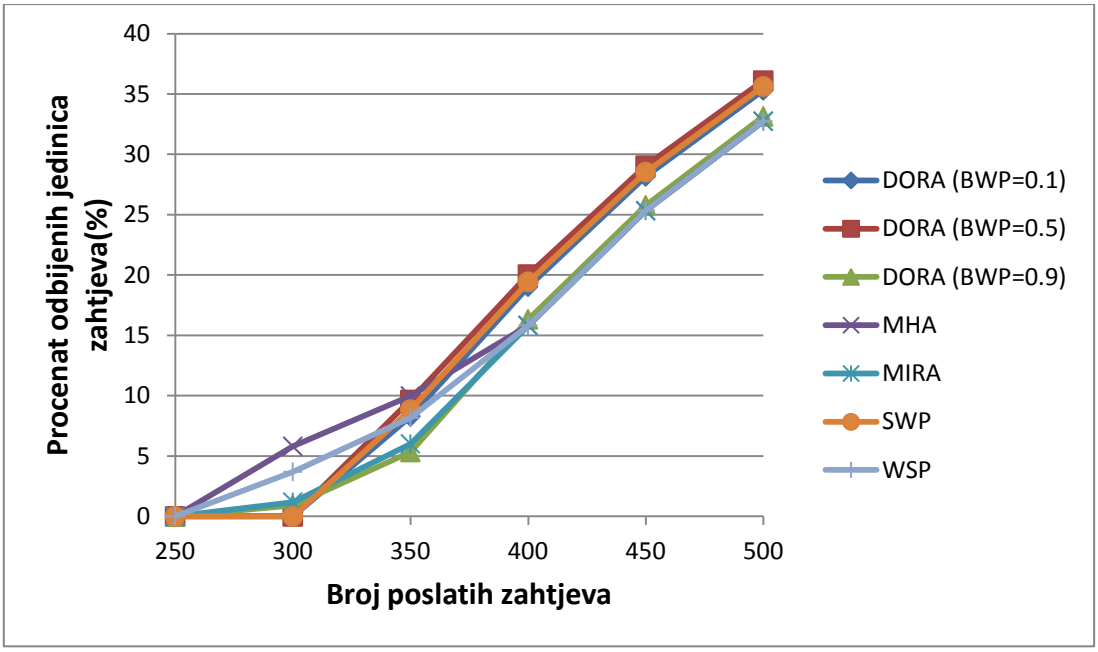
U drugom eksperimentu izvršeno je poređenje različitih QoS algoritama u pogledu broja odbijenih jedinica zahtjeva tokom simulacije. Na Slici 15. prikazani su dobijeni rezultati za MIRA topologiju. Kao što je očekivano, MHA algoritam počinje prije ostalih sa odbijanjem zahtjeva. Ovo je posledica nebalansiranog opterećenja mreže, koje dovodi do brzog trošenja resursa pojedinih linkova i rezultira brzo situacijom da ni jedna ruta između izvornog i destinacionog čvora nije upotrebljiva. Dobijeni rezultati pokazuju da algoritmi koji akcenat stavljaju na balasiranje opterećenja (u ovom slučaju SWP) pokazuju dobre performanse u uslovima slabog saobraćajnog opterećenja. Međutim, s obzirom da izbjegavanjem linkova na kratkim putanjama SWP troši dosta mrežnih resursa, sa porastom opterećenja njegove performanse naglo opadaju. Sa druge strane algoritmi koji favorizuju kraće rute (MHA i WSP) imaju najlošije razultate pri malom opterećenju, jer najbrže potroše resurse na nekim

od kritičnih linkova, ali pojedini djelovi mreže ostaju neiskorišćeni i mogu se koristiti kasnije za rutiranje zahtjeva imeđu parova izvor-destinacija čija konektivnost nije ugrožena. Algoritmi koji uzimaju u obzir "interferenciju" između parova pokazali su se najjefikasnijim u pogledu broja odbijenih zahtjeva. Izuzetak je DORA algoritam u slučaju kada je *BWP* parametar postavljen na malu vrijednost (0.1). Tada DORA akcenat stavlja na izbjegavanje kritičnih linkova, tj. linkova za koje procjenjuje da će u budućnosti najviše učestvovati u prenosu saobraćaja različitih parova. Aspekt slobodnog propusnog opsega na linkovima je zanemaren. Sa druge strane MIRA algoritam koji je potpuno fokusiran samo na izbjegavanje "interferencije" pokazuje znatno bolje rezultate. Razlog je što je kod MIRA algoritma koncept interferencije mnogo preciznije definisan, tako da se linkovi smatraju kritičnim samo ukoliko uspostavljanje QoS tunela preko njih dovodi do smanjenja vrijednosti maksimalnog toka za neki od parova izvor-destinacija. DORA u cilju bržeg izvršavanja aproksimira ovaj koncept, pa se dešava da se neki linkovi proglase više kritičnim nego što to zapravo jesu. Takođe, kritičnim se proglašavaju i linkovi koji ne predstavljaju usko grlo u mreži, jer se prva faza algortima izvršava samo prilikom promjene u topologiji. Zbog toga težinski faktori ne predstavljaju dobar indikator njihove važnosti za buduće zahtjeve. Pri velikoj vrijednosti *BWP* parametra, DORA uvijek daje najbolje rezultate. To ukazuje na značaj dinamičkog rutiranja, ali i važnost korišćenja informacije o slobodnom kapacitetu prilikom definisanja težinskih faktora linkova. U ovakvoj konfiguraciji DORA na osnovu informacija o položaju ulazno-izlaznih parova vrši estimaciju kritičnosti linkova za buduće zahtjeve, ali iste uzima sa rezervom tako što uvijek više štiti one linkove koji su deficitarni sa slobodnim propusnim opsegom.



Slika 15: Procenat odbijenih jedinica zahtjeva tokom simulacije na MIRA topologiji

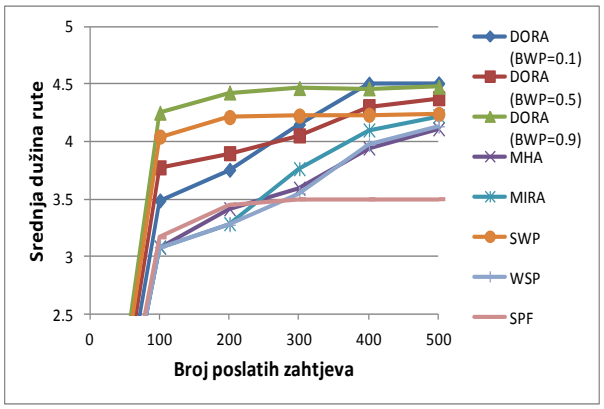
Rezultati simulacije na proširenoj Verizon topologiji prikazani su na Slici 16. I u ovom scenariju performanse algoritama su u sličnom odnosu kao kod MIRA topologije, tako da se mogu uočiti određene pravilnosti u njihovom ponašanju.



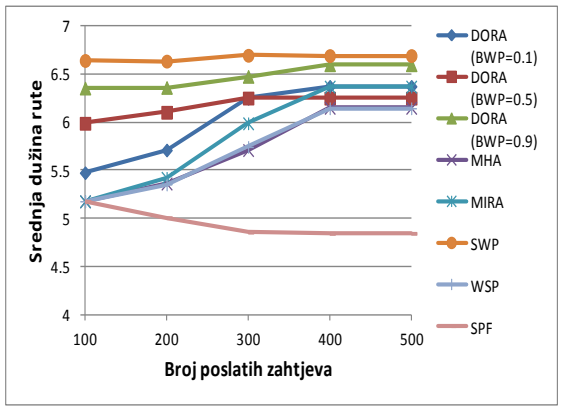
Slika 16: Procenat odbijenih jedinica zahtjeva tokom simulacije na Verizon topologiji

Na Slikama 17 i 18 prikazana je zavisnost između srednje dužine rute prihvaćenih zahtjeva (u broju hopova) i broja poslatih zahtjeva za MIRA i Verizon topologiju respekitvno. Dužina rute je važan indikator performansi jer se pored potrošnje resursa odražava i na kašnjenje saobraćaja. Putanje sa većim brojem hopova unose dodatno propagaciono kašnjenje i kašnjenje u redovima čekanja. Sa grafika se može vidjeti da se od rešenja za inženjering saobraćaja MHA i WSP algoritmi ponašaju najbolje u tom pogledu, što je i očekivano. Sa druge strane rešenja koja su fokusirana na balansiranje opterećenja biraju duže putanje ukoliko one imaju više slobodnog propusnog opsega na raspolaganju. Bitno je uočiti da je DORA algoritam (*BWP*=0.9), koji se pokazao najboljim u pogledu vjerovatnoće blokiranja zahtjeva, po ovom kriterijumu uvijek najgori ili među najgorima. Iako su s obzirom na njegovu dominantnu fokusiranost na balansiranje opterećenja duže rute očekivane, interesantno je da su njegove rute često duže i od ruta SWP algoritma koji uvijek traži "najširu" putanju. DORA u definiciji težinskog faktora koristi recipočnu vrijednost slobodnog kapaciteta na linku, što je ipak kompromisno rešenje između balansiranja opterećenja i rutiranja najkraćom putanjom. Međutim, DORA pored zagušenih linkova izbjegava i linkove za koje procjenjuje da su od najveće važnosti za prihvatanje budućih zahtjeva, zbog čega su putanje prilično dugačke čak i pri malom saobraćajnom opterećenju. Sa grafika se može primijetiti da je ovaj problem manje izražen pri manjim vrijednostim *BWP* parametra, ali je to ostvareno na račun manjeg broja prihvaćenih QoS zahtjeva.

Iz analize se može zaključiti da algoritmi za inženjering saobraćaja koji su optimalni sa aspekta NSP provajdera (zbog male vjerovatnoće blokiranja zahtjeva), nisu prihvatljivi za provajdere servisa osjetljivih na kašnjenje, jer im se ni pri malom opterećenju mreže ne mogu garantovati prihvatljive performanse. Za servise ovog tipa, neophodni su algoritmi koji će pored propusnost i kašnjenje uzeti kao ograničenje prilikom rutiranja.



Slika 17: Srednja dužina rute na MIRA topologiji



Slika 18: Srednja dužina rute na Verizon topologiji

4.4 QoS ALGORITMI ZA GARANCIJU KAŠNJENJA I PROPUSNOSTI

Usled rapidnog razvoja velikog broja *real-time* aplikacija nametnula se potreba za algoritmima rutiranja koji mogu da pruže garanciju kašnjenja u mreži. Pod pretpostavkom da su kašnjenja na likovima poznata, ovaj problem se jednostavno rešava SPF algoritmom. Sa druge strane, uspostavljanje QoS tunela u *backbone* mreži podrazumijeva i rezervaciju propusnog opsega za agregirane tokove podataka. Problem pronalaženja rute koja zadovoljava zahtjev u pogledu propusnosti i kašnjenja se i dalje može riješiti u polinomijalnom vremenu. Tipično, rutiranje se obavlja u dva koraka. Prvo se svi linkovi koji nemaju dovoljno resursa uklone iz grafa. Zatim se koristi Dijkstra ili Bellman Ford algoritam za pronalaženje najkraće putanje u pogledu kašnjenja. Ukoliko je kašnjenje na toj putanji manje od zahtijevanog, moguće je pružiti traženi kvalitet servisa. Ovaj algoritam je inicijalno predložen u [74], i u nastavku rada spominjaće se kao MDA (*Minimum Delay Algorithm*) algoritam.

S obzirom da se u ovom magistarskom radu razmatra scenario uspostavljanja saobraćajnih tunela u *backbone* mreži, pored pružanja zahtijevanog kvaliteta servisa iz ugla NSP provajdera važno je da algoritam koji se izvršava na SDN kontroleru efikasno koristi mrežne resurse i omogući prihvatanje što većeg broja zahtjeva. MDA algoritam nije dizajniran sa tim ciljem i ne može se smatrati tehnikom za inženjering saobraćaja. U prethodnoj sekciji su predstavljena tri ključna pristupa inženjeringu saobraćaja a to su: smanjenje potrošnje resursa izborom kraćih putanja, balansiranje opterećenja i izbjegavanje interferencije između parova izvor-destinacija. MDA algoritam ne slijedi striktno ni jedan od njih. Od TE (*Traffic Engineering*) agoritama za garanciju kašnjenja i propusnosti koji su predloženi u literaturi po performasama ističe se MDWCRA (*Maximum Delay-Weighted Capacity Routing Algorithm*) algoritam [75]. Slično MIRA algoritmu, i MDWCRA je baziran na ideji minimizacije interferencije između ulazno-izlaznih parova, s tim što je ovdje koncept interferencije interpretiran na drugačiji način. Smatra se da zahtjev stvara interferenciju za ostale parove izvor-destinacija ukoliko se rutira nekim od *bottleneck* linkova njihovih najkraćih putanja. Ovi linkovi smatraju se kritičnim, i izbjegavaju se prilikom rutiranja tako

što im se dodjeljuje težinski faktor koji je rastuća funkcija njihovog "nivoa kritičnosti". Dakle, problem računanja težinskih faktora svodi se na problem pronalaženja seta kritičnih linkova za sve parove izvor-destinacija. Algoritam se izvršava u 6 koraka:

1. Računa se set nepreklapajućih putanja sa najmanjim kašnjenjem za svaki par izvor-destinacija (*s*,*t*).
2. Određuje se set kritičnih linkova (C_{st}) za vaki par (*s*,*t*). Kritični linkovi su *bottleneck* linkovi izračunatih putanja.
3. Računaju se težinski faktori linkova:

$$w(l) = \sum_{(s,t):\, l \in C_{st}} \alpha_{st} \tag{7}$$

4. Eliminišu se svi linkovi koji nemaju dovoljno slobodnog propusnog opsega.
5. EDSP (*Extended Dijskstra Shortest Path*) ili EBF (*Extended Bellman Ford*) algoritmom [76] računa se putanja sa najmanjim težinskim faktorom koja zadovolja zahtjev u pogledu kašnjenja.
6. Zahtjev se rutira izračunatom putanjom i ažuriraju se vrijednosti slobodnog kapaciteta u mreži.

Predložene su tri različite definicije težinskog faktora koje su određene parametrom α_{st} :

1. Težinski faktor predstavlja broj parova za koji je link *l* kritičan:

$$w(l) = \sum_{(s,t):\, l \in C_{st}} 1 \quad , \; za \;\; \alpha_{st} = 1 \tag{8}$$

2. Težinski faktor je obrnuto proporcionalan ukupnom kašnjenju kritične putanje:

$$w(l) = \sum_{(s,t):\, l \in C_{st}} \frac{1}{D_{st}} \quad , \; za \;\; \alpha_{st} = \frac{1}{D_{st}} \tag{9}$$

3. Težinski faktor je obrnuto proporcionalan proizvodu propusnog opsega i kašnjenja kritične putanje:

$$w(l) = \sum_{(s,t):\, l \in C_{st}} \frac{1}{B_{st} \cdot D_{st}} \quad , \; za \;\; \alpha_{st} = \frac{1}{B_{st} \cdot D_{st}} \tag{10}$$

Najveći nedostatak ovog algoritma je njegova kompleksnost. Prilikom određivanja ruta postoje dva ograničenja: u pogledu propusnosti i pogledu kašnjenja, a dodatno se optimizuje težinski faktor (aditivna metrika) putanje. Ovaj problem se nakon eliminisanja linkova koji nemaju dovoljno resursa na raspolaganju svodi na problem ograničenja i optimizacije rute, koji je *NP-complete* (Sekcija 4.1.4). EDSP i EBF algoritmi pojednostavljuju ovaj problem pod pretpostavkom diskretizacije kašnjenja na linkovima. Usled diskretizacije mogući broj vrijednosti kašnjenja na ruti se smanjuje što omogućava rešavanje problema u polinomijalnom vremenu. Međutim efikasnost ovih tehnika zavisi od greške koja se unosi postupkom diskretizacije, koja je raste sa dužinom rute. Mala greška zahtijeva diskretizaciju finijeg nivoa, što povlači veliko vrijeme izvršavanja algoritma i nije prihvatljivo za

dinamička okruženja kao što su *backbone* mreže u kojima nekoliko SDN kontrolera obrađuje sve zahtjeve.

S obzirom da i MHA, SWP, WSP, MIRA i DORA algoritmi moraju biti upotpunjeni EDSP ili nekim sličnim algoritmom da bi garantovali kašnjenje, Algoritmom 2 je opisan pseudo-kod EDSP implementacije koja je korišćena u simulacijama.

Pretpostavlja se da su poznat težinski faktori ($w_1(l)$, $\forall\ l \in L$) i kašnjenja ($d(l), \forall\ l \in L$) na linkovima, i da konekcija zahtjeva kašnjenje manje ili jednako od D_{max}. U cilju bržeg pronalaženja rešenja u [76] predložena je diskretizacija mogućih vrijednosti kašnjenja. Diskretizacija podrazumjeva zamjenu metrike kašnjenja na linku novom metrikom:

$$w_2(l) = \left\lceil \frac{d(l) \cdot x}{D_{\max}} \right\rceil \tag{11}$$

gdje je x pozitivan cijeli broj. Problem se na ovaj način transformiše u pronalaženje rute sa najmanjim težinskim faktorom $w_1(P)$, koja zadovoljava uslov: $w_2(P) < x$.

Za svaki čvor grafa $v \in N$, i cijeli broj $k \in [0...x]$ održava se promjenjiva *dist[v,k]* koja predstavlja estimaciju težinskog faktora w_1 optimalne putanje između izvorišnog čvora i čvora v, za koju važi da je $w_2(P)=k$. Inicijalno *dist[v,k]* se postavlja na beskonačno veliku vrijednost. Tokom izvršavanja EDSP algoritam sve preciznije estimira *dist[v,k]*, dok eventualno ne odredi zaista optimalnu vrijednost.

Na kraju izvršavanja EDSP algoritma dobija se set vrijednosti *dist[v,k]*, za svako $v \in N$ i svako $k \in [0...x]$. Promjenjiva *pret[v,k]* čuva informaciju o prethodnom čvoru na odgovarajućoj putanji. Na ovaj način ruta se može rekonstruisati polazeći od destinacionog čvora, kroz sve međučvorove do izvora. Uvedene su dvije dodatne varijable S i Q koje imaju sledeće značenje:

$$S = \left\{ \langle v,k \rangle \mid dist[v,k] = dist_{optimalno}[v,k], v \in N, k \in [0...x] \right\} \tag{12}$$

$$Q = \left\{ \langle v,k \rangle \mid dist[v,k] > dist_{optimalno}[v,k], v \in N, k \in [0...x] \right\} \tag{13}$$

Inicijalno S je prazan skup dok je $Q=\{(v,k)\ \forall v \in N,\ \forall k \in [0...x]\}$. U svakoj iteraciji *while* petlje (linije 26-31 Algoritma 2) jedan element se iz skupa Q prebacuje u S, i prilagođava se estimirani težinski faktor za taj element u funkciji *Relax(u,k,v).* Algoritam se završava kada se Q isprazni.

Složenost ovog algoritma je $O(x^2N^2)$ i dominantno određuje kompleksnost MDWCRA algoritma, kao i modifikacija MHA, WSP, SWP, MIRA i DORA algoritama sa podrškom za garanciju kašnjenja.

Algoritam 2: EDSP

```
1:# preth[v, k]- prethodni čvor na optimalnoj ruti do čvora v sa kašnjenjem k
2:# w1 - težinski faktor linka, w2 - modifikovana metrika kašnjenja na linku
3:# dist[v, k]- estimirani težinski faktor (w1) rute do čvora v sa kašnjenjem k
4:
5:  function: inicijalizacija(G,izv):
6:      for v in N[G] do:
7:        for i in [0... x] do:
8:            dist[v,i] = ∞
9:            preth[v,i] = Null
10:     for i in [0... x] do:
11:           dist[izv,i] = 0
12: end function
13:
14: function Relax(u,k,v):
15:     k'=k+w2(l_u,v)
16:     if k'≤ x then:
17:       if dist[v,k'] > dist[u,k] + w1(l_u,v) then:
18:           dist[v,k'] = dist[u,k] + w1(l_u,v)
19:           preth[v, k'] = u
20: end function
21:
22: function EDSP(G,izv):
23:     inicijalizacija(G,izv)
24:     S=[ ]
25:     Q={(u,v) | v ∈ N[G], k ∈ [0... x]}
26:     while Q ≠ {} do:
27:       pronađi (u,k) ∈ Q tako da je dist[u,k]= Min {dist [u',k']} | ∀(u',k') ∈ Q}
28:       Q = Q - {dist[u,k]}
29:       S = S + {dist[u,k]}
30:       for v in u.susjedi() do:
31:           Relax(u,k,v)
32: end function
```

4.5 NOVI ALGORITAM ZA GARANCIJU KVALITETA SERVISA

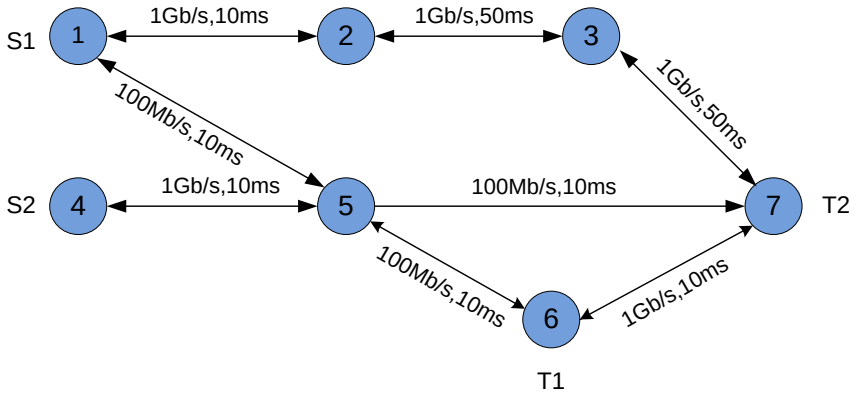
S obzirom da je ključni nedostatak algoritama za garanciju kašnjenja i propusnosti njihova kompleksnost, u ovoj sekciji predložen je jednostavniji model QoS okruženja. Umjesto scenarija u kojem korisnik sa NSP provajderom može da pregovara bilo koju vrijednost kašnjenja u mreži, razmatran je scenario kada se QoS zahtjevi klasifikuju u konačan broj kategorija. U osnovnom slučaju to su:

1. DS (*Delay Sensitive*) - koja ne toleriše kašnjenje veće od $D_{max,}$ i
2. NDS (*Non-Delay Sensitve*) - koja nije osjetljiva na kašnjenje (npr. saobraćaj FTP, Email, Backup ili Web aplikacija)

Ovakav pristup eliminiše potrebu za kompleksnim algoritmima kao što su EDSP i EBF. Ideja je da SDN kontroler pri inicijalizaciji mreže izračuna set putanja sa kašnjenjem koja su prihvatljiva za DS saobraćaj, a da pri dolasku DS zahtjeva relativno brzo na osnovu određenog kriterijuma bira jednu od njih. Po potrebi DS zahtjevi se mogu diferencirati u više potklasa sa različitim ograničenjima u pogledu kašnjenja, tako da je u praksi sa stanovišta korisnika ovaj model prihvatljiv.

S obzirom da MDWCRA predstavlja *state-of-the-art* rešenje za garanciju kašnjenja i propusnosti, predloženi algoritam baziran je na sličnoj ideji, s tim što pokušava da prevaziđe neke njegove nedostatke. Pretpostavlja se da su parovi ulazno-izlanih čvorova u mreži poznati, kao i propagaciona kašnjenja i slobodni kapacitet na linkovima. Za razliku od MDWCRA algoritma koji je dizajniran za rutiranje saobraćaja sa striktnim zahtjevima u pogledu kašnjenja, ovdje je razmatran realniji scenario u kojem nije sav saobraćaj osjetljiv na kašnjenje. Definisane su odvojene funkcije za rutiranje DS i NDS saobraćaja. Pseudo-kod algoritma predstavljen je Algoritmom 3.

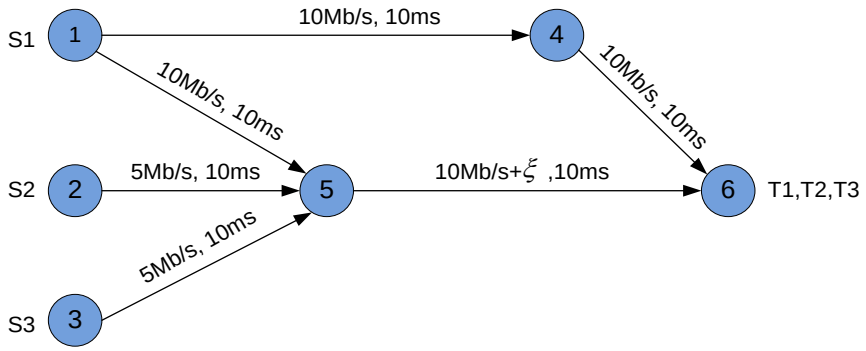
U cilju bržeg izvršavanja, algoritam je podijeljen na *offline* i *online* fazu. U *offline* fazi vrši se najzahtjevnija operacija, a to je računanje ruta koje mogu da ispune zahtjeve u pogledu kašnjenja DS saobraćaja. Ova operacija je bazirana na Yen-ovom algoritmu za računanje *K* najkraćih putanja između izvorišnog i destinacionog čvora grafa [77]. Za svaki od parova računa se set od *K* putanja, pri čemu je dodato ograničenje da se algoritam prekida kada pronađe putanju sa kašnjenjem većim od *Dmax*. Treba naglasiti da Yen-ov algoritam kao rezultat daje zaista najkraće (u pogledu kašnjenja) putanje u mreži. Ovo je bitna razlika u odnosu na MDWCRA algoritam, koji za svaki od parova računa samo nepreklapajuće putanje. Problem drugog pristupa je što se često dešava da se linkovi na putanjama koje su potpuno beskorisne za DS saobraćaj proglase kritičnim, dok se sa druge strane ne štite linkovi nekih kratkih putanja koje nisu ni prepoznate kao bitne zbog ograničenja nepreklapanja. Primjer takve situacije prikazan je na Slici 19. Pod pretpostavkom da je *Dmax*=100ms, samo link (5,7) će se smatrati kritičnim za par S2-T2, jer je on *bottleneck* na ruti 4-5-7. Za par S1-T1 MDWCRA proglasiće kritičnim linkove (1,5) i (5-6), ali i linkove (1,2), (2,3), (3,7) i (7,6), iako je kašnjenje na putanji 1-2-3-7-6 120ms. Pri ovakvoj postavci stvari, NDS saobraćaj između S1 i T1 nepotrebno će se prenositi rutom 1-5-6, i trošiti resurse koji bi se mogli u budućnosti koristiti za DS saobraćaj oba para.



Slika 19: Primjer topologije

Pri malim vrijednostima *K* parametra moguće je da set putanja izračunatih Yen-ovim algoritmom ne obuhvati sve nepreklapajuće putanje sa kašnjenjem manjim od D_{max}. S obzirom na njihov značaj u prenosu DS saobraćaja, predloženi algoritam uvijek računa nepreklapajuće rute koje zadovoljavaju ograničenje u pogledu kašnjenja, i dodaje ih setu ruta izračunatih Yen-ovim algoritmom u slučaju da se u njemu već ne nalaze.

Iako se DS putanje računaju u *offline* fazi, kritični linkovi moraju se odrediti *online* jer se potencijalno mogu promijeniti nakon rutiranja svakog zahtjeva (linije 14-29). Definicija kritičnih linkova izmijenjena je u odnosu na MDWCRA algoritam. Kod MDWCRA algoritma link se smatra kritičnim za određeni par izvor-destinacija ukoliko predstavlja usko grlo na nekoj od njegovih najkraćih putanja (nepreklapajućih). Osnovna ideja je minimizacija interferencije između parova, tj. spriječiti da rutiranje zahtjeva utiče na maksimalnu količnu saobraćaja koju ostali parovi mogu da prenesu svojim najkraćim putanjama. *Bottleneck* linkovi se biraju kao kritični jer rutiranje jedne jedinice saobraćaja preko ovih linkova sigurno stvara "interferenciju". To takođe znači da rutiranje jedinice saobraćaja preko "nekritičnih" linkova nema negativnih efekata. Međutim, ukoliko se kapacitet nekritičnog linka nekog para smanji za više od jedne jedinice, moguće je da do "interferencije" ipak dođe. Na Slici 20. ilustrovan je primjer ovog problema. Pretpostavlja se da S1 šalje zahtjev za uspostavljanjem 10Mb/s saobraćajnog tunela i da nijedna ruta nije problematična u pogledu kašnjenja. Pri prikazanim uslovima, MDWCRA će za par S1-T1 proglasiti kritičnim linkove (1,4), (4,6) i (1,5), dok su za S2-T2 i S3-T3 kritični linkovi (2,5) i (3,5) redom. Tunel će se uspostaviti preko rute 1-5-6 koja obuhvata samo jedan kritičan link. Ovo ne predstavlja optimalno rešenje jer se QoS zahtjevi od S2 i S3 više ne mogu prihvatati. Iz tog razloga, u predloženom algoritmu kritičnim se smatraju svi linkovi *K*-najkraćih putanja čiji je slobodni kapacitet manji od sume *bottleneck* kapaciteta putanje i pristiglog zahtjeva u pogledu propusnosti (linija 20). S obzirom da sada jedan isti link može da bude proglašen više puta kritičnim za isti par (jer se ne radi o nepreklapajućim putanjama), rizično je da njegov težinski faktor ne postane nepotrebno previše veliki. Da bi se izbjegla takva situacija kada je link pod malim saobraćajnim opterećenjem, svaki put kada se utvrdi da je link kritičan za neku putanju, težinski faktor se uvećava recipročno njegovom slobodnom kapacitetu (linija 21). Treba još primijetiti da u slučaju obrade DS zahtjeva vrijednosti težinskih faktora ne zavise od kritičnih linkova njegovog izvora i destinacije (linija 16). Samo NDS saobraćaj para mora da izbjegava ove linkove.



Slika 20: Problematičan izbor kritičnih linkova kod MDWCRA algoritma (ξ je vrlo mala vrijednost)

Konkretno u ilustrovanom primjeru, ukoliko S1 ne šalje odmah 10Mb/s zahtjev već više veoma malih zahtjeva iste zbirne propusnosti, problem i dalje postoji. Sve dok se slobodni kapacitet linka (5,6) ne približi vrijednosti od 5Mb/s, on se neće smatrati kritičnim. Ovo ukazuje na potrebu za postavljanjem još nižeg kriterijuma prilikom određivanja kritičnih linkova. Međutim, takav pristup ima svoje nedostatke. Veliki broj linkova proglašava se kritičnim pa se na kraju težinski faktori približavaju po vrijednostima. Sa druge strane, zbog malog broja nekritičnih linkova, NDS saobraćaj bira jako duge putanje i samim tim neefikasno troši mrežne resurse.

Sledeći bitan nedostatak MDWCRA algoritma je što prilikom donošenja odluka ne razmatra slobodni kapacitet na linkovima koji nisu kritični. Ovo je bitno sa aspekta rutiranja NDS saobraćaja jer omogućava balansiranje opterećenja u meži. Stoga je kod predloženog algoritma ukupni težinski faktor definisan iz dva dijela, slično kao što je urađeno kod DORA algoritma za garanciju propusnosti (linija 28). Parametar *BWP* podešen je na malu vrijednost (0.1) jer se želi akcenat staviti na minimizaciju interferencije između parova, a slobodni kapacitet linkova koristiti samo za pravljenje finijih razlika između podjednako kritičnih ruta.

Kada se težinski faktori odrede, zavisno od toga da li je riječ o NDS ili DS zahtjevu izvršava se Dijkstra algoritam ili *DS_ruta()* fukcija. U prvom slučaju računa se samo najkraća putanja na težiranom grafu. U drugom slučaju, od putanja izračunatiih u *offline* fazi bira se ona najmanjim težinskim faktorom. Ukoliko ni jedna od njih nema dovoljno resursa na raspolaganju rutiranje se vrši MDA algoritmom, čija je složenost jednaka složenosti SPF algoritma.

Offline faza algoritma je dominantan nosilac složenosti i njeno vrijeme izvršavanje je $O(pKN^3)$ [77], gdje je p broj parova ulazno-izlaznih čvorova. Međutim, kao što je već spomenuto, ova faza se izvršava samo prilikom inicijalizacije ili promjene u topologiji. Događaji kao što su ispadi linkova ili uređaja nisu toliko kritični, jer zahtijevaju samo eliminaciju već izračunatih DS ruta.

4.6 POREĐENJE PERFORMANSI

U ovoj sekciji ispitane su performanse predloženog algortima na MIRA topologiji (Slika 11). Kašnjenja na linkovima su uniformno raspoređena u opsegu od 1ms do 50 ms. Pretpostavljen je scenario sa četiri para ulazno-izlaznih čvorova koji je često razmatran u literaturi [9][10][75]. Svaki od parova generiše sa jednakom vjerovatnoćom QoS zahtjeve koji su zadati u formatu (*s,t,B,D*), gdje *s* i *t* predstavljaju izvorni i destinacioni čvor, a *B* i *D* ograničenja u pogledu propusnosti i kašnjenja. Zahtjevi u pogledu propusnosti uniformno uzmimaju jednu iz seta vrijednosti [10,20,30,40].

Osim MDA i MDWCRA algoritama, koji su dizajnirani za garanaciju kašnjenja i propusnosti, u simulacijama razmatrane su i performanse MHA, SWP, WSP, MIRA i DORA algoritama, koji su za svrhu sprovedene analize nadograđeni EDSP algoritmom. Modifikovane verzije ovih algoritama, koje pored propusnosti garantuju i kašnjenje na

izabranim rutama, u nastavku biće označene sa D_MHA, D_SWP, D_WSP, D_MIRA i D_DORA redom. U cilju dobijanja što obuhvatnijeg uvida u ponašanje različitih algoritama u različitim situacijama analizirano je više scenarija.

Algoritam 3: Novi QoS algoritam

```
1:# w(l) - težinski faktor linka l, id_parovi - lista parova izvor-destinacija
2:# B - zahtjev u pogledu propusnosti, ds_ind - indikator DS saobraćaja
3:# izv - izvorni čvor, dest - destinacioni čvor, opt_ruta - optimalna ruta
4:# G - graf sa linkovima koji zadovoljavaju zahtjev u pogledu propusnosti
5:# ruta.težina()- težinski faktor rute
6:
7:  function: offline_faza(G)
8:    for (izv,dest) in id_parovi do:
9:       #računanje K-najkraćih putanja
10:      ds_rute[(izv,dest)] = Yenov_algoritam(G,izv,dest,Dmax,K)
11:      ds_rute[(izv,dest)] += nepreklapajuće_DS_rute(G,izv,dst,Dmax)
12: end function
13:
14: function online_faza(izv,dest,B,ds_ind):
15:   for (i,d) in id_parovi do:
16:      if ds_ind == True and (izv1,dst1) == (izv,dest) then:  continue
17:      for put in ds_rute[(i,d)] do:
18:         if propusni_opseg(put) < B then: continue
19:         for l in put.linkovi() do:
20:            if sl_kapacitet(l) < propusni_opseg(put)+B then:    #link je kritičan
21:               w1[l]+=1/ sl_kapacitet(l)
22:  normalizacija(w1) #normalizuje vrijednosti u opseg od 1 do 10
23:  for link in L(G) do:
24:     if sl_kapacitet(link) > B then:
25:        w2[link]=1/sl_kapacitet(link)
26:  normalizacija(w2)
27:  for link in L(G) do:
28:        w[link]=w1(link)+BWP*w2(link)
29: end function
30:
31: function  kalkulacija_ruta(G,izv,dest,B,ds_ind):
32:  online_faza(izv,dest,B,ds_ind)
33:  if ds_ind == 1 then:
34:     return DS_ruta(izv,dest,w)
35:  else:
36:     return Dijkstra(G,izv,dest,w)
37: end function
38:
39: function DS_ruta(izv,dest,w):
40:  opt_ruta=Null
41:  for ruta in ds_putevi[izv,dst] do:
42:     if sl_kapacitet(ruta)>=B then:
43:        if opt_ruta == Null or ruta.težina()<opt_ruta.težina():
44:           opt_ruta=ruta
45:  if opt_ruta == Null:
46:     opt_ruta=MDA(G,izv,dest,w)
47:  return opt_ruta
48: end function
```

4.6.1 PROCENAT ODBIJENIH JEDINICA SAOBRAĆAJA

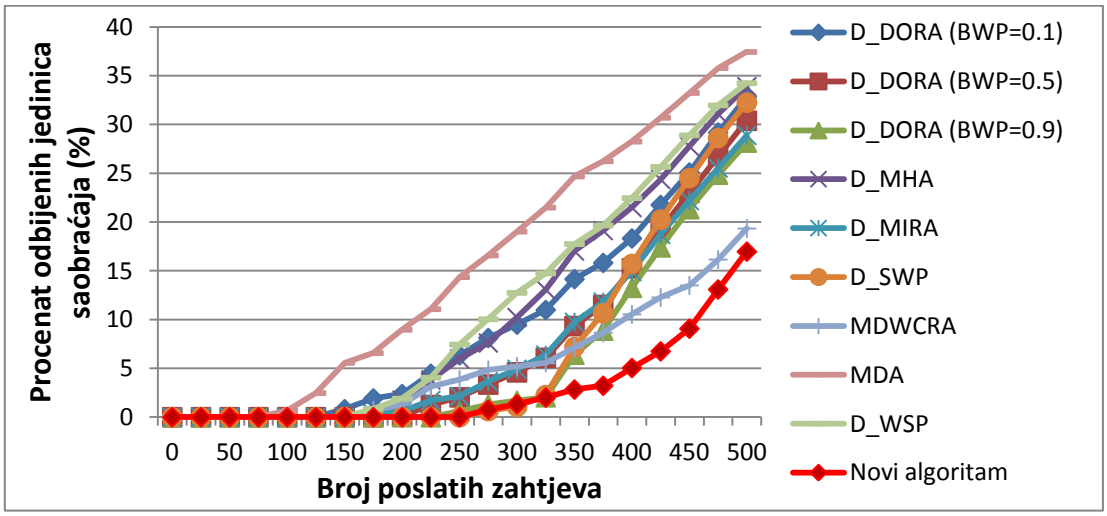
U prvom scenariju generisano je 500 zahtjeva, od kojih se 60% odnosilo na DS saobraćaj, a 40 % na NDS saobraćaj. Vrijednost maksimalnog dozvoljenog kašnjenja za DS saobraćaj inicijalno je podešena na 110 ms. Na Slikama 21, 22 i 23 prikazani su rezultati u pogledu procenta odbijenih jedinica ukupnog generisanog saobraćaja, i različitih saobraćajnih klasa pojedinačno. Vrijednosti predstavljene na graficima su srednje vrijednosti rezultata nakon pet uzastopnih simulacija sa različitim distibucijama kašnjenja na linkovima. Može se uočiti da MDWCRA i njegova modifikacija koja je predložena u radu značajno odstupaju po performansama u odnosu na druge razmatrane algoritme. Pri manjem broju zahtjeva performanse svih algoritama su bliske jer se zahtjevi prihvataju sa velikom vjerovatnoćom u uslovima slabog saobraćajnog opterećenja. Pri većem opterećenju razlike postaju značajne. Najlošiji rezultati dobijeni su za MDA algoritam. Ovaj algoritam koristi putanju sa najmanjim kašnjenjem između ulaznog i izlaznog čvora za upostavljanje oba tipa konekcija. Tek kada na njoj ponestane slobodnog kapaciteta bira se alternativna ruta. Putanja sa najmanjim kašnjenjem je očigledno najbolje rešenje za pružanje garancija u pogledu kašnjenja. Međutim, ovaj pristup smanjuje vjerovatnoću prihvatanja budućih DS zahtjeva. Takođe, putanja sa najmanjim kašnjenjem između jednog para ulazno-izlaznih čvorova obično sadrži linkove sa malim kašnjenjem. Samim tim velika je vjerovatnoća da ti linkovi pripadaju optimalnim putanjama ostalih parova ulazno-izlaznih čvorova u mreži. Stoga, rutiranje zahtjeva MDA algoritmom dovodi do brzog trošenja resursa na tim kritičnim putanjama.

D_MHA se pokazao kao drugi najgori u najvećem broju simulacija. Ovaj algoritam bira alternativnu putanju tek kada ponestane slobodnog kapaciteta na najkraćoj putanji između izvornog i destinacionog čvora. Kao ni kod MDA, odluke o rutiranju nisu bazirane na lokaciji ulazno-izlaznih čvorova u mreži. Brzo zagušenje linkova na najkraćim putanjama posledično dovodi do čestog blokiranja zahtjeva parova za koje su ti linkovi bitni, naročito sa aspekta rutiranja DS saobraćaja. Sa Slike 23 može se primijetiti da D_MHA favorizuje NDS saobraćaj. Međutim, znatno manji procenat odbijenih NDS zahtjeva dominantno je posledica velikog broja odbijenih DS zahtjeva, usled čega više resursa ostaje slobodno u mreži. D_WSP algoritam baziran je na sličnom principu, i u zavisnosti od distibucije kašnjenja na linkovima tokom simulacija njegove performanse su neznatno bolje ili lošije od D_MHA algoritma.

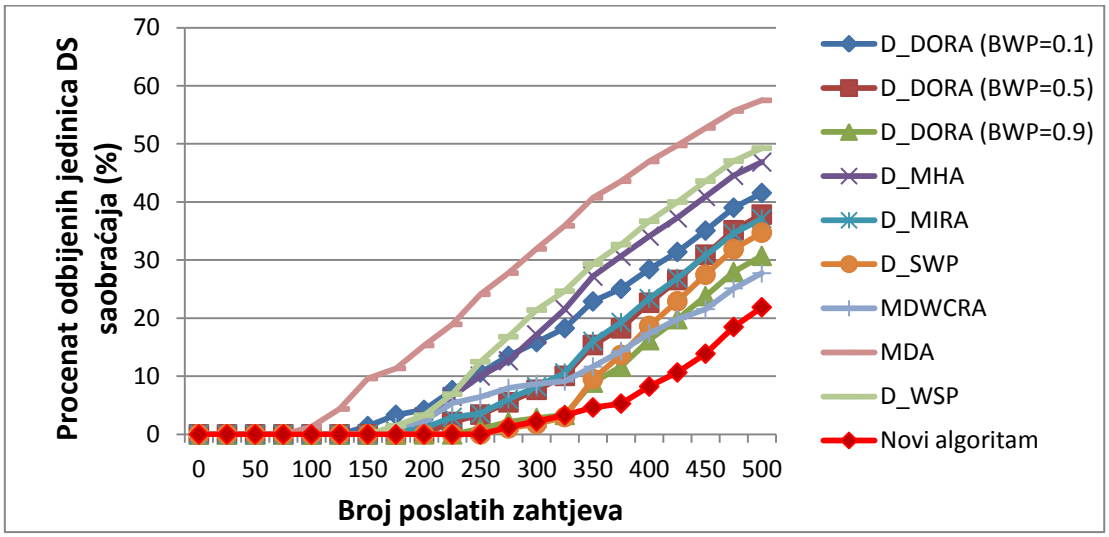
Procenat odbijenih jedinica saobraćaja kod D_MIRA algoritma je takođe veliki, iako su odluke o rutiranju bazirane na informacijama o položaju ulazno-izlaznih čvorova. Razlog je taj što kritičnost linka zavisi samo od slobodnog propusnog opsega. S obzirom da linkovi kritični u pogledu propusnog opsega nisu obavezno kritični u pogledu kašnjenja, njihovom zaštitom postižu se uglavnom negativni efekti kada je DS saobraćaj u pitanju.

D_SWP algoritam se u uslovima slabog opterećenja ponaša znatno bolje od prethodno navedenih algoritama. Forsiranjem putanja sa najviše slobodnog propusnog opsega D_SWP efikasno balansira opterećenje linkova, što je rezultiralo manjom vjerovatnoćom blokiranja

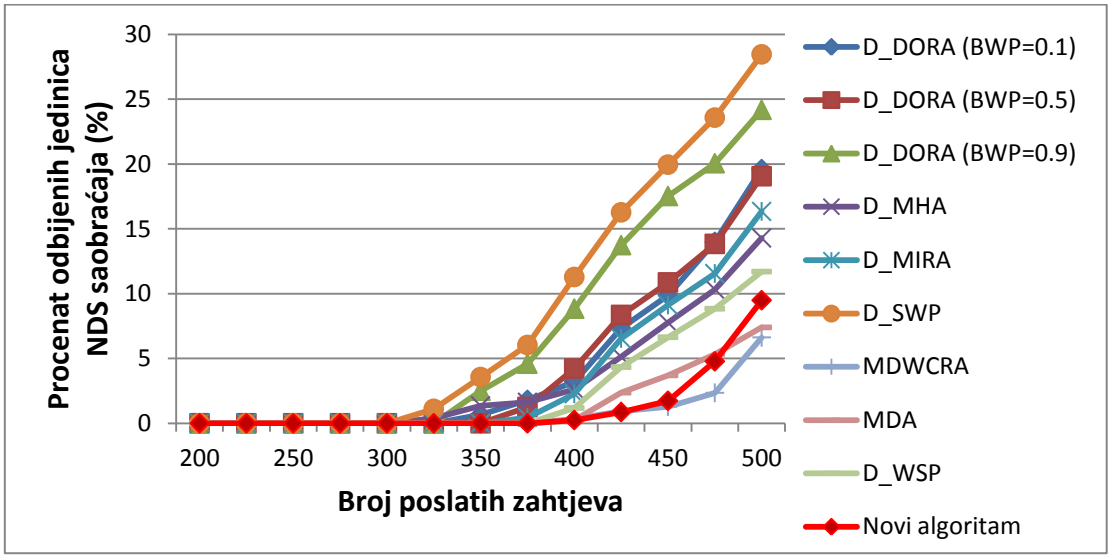
DS zahtjeva. Pri većem opterećenju performanse se naglo pogoršavaju, prije svega zbog rutiranja NDS soabraćaja veoma dugačkim putanjama koje troše ogromne resurse.



Slika 21: Procenat odbijenih jedinica saobraćaja, Dmax=110ms



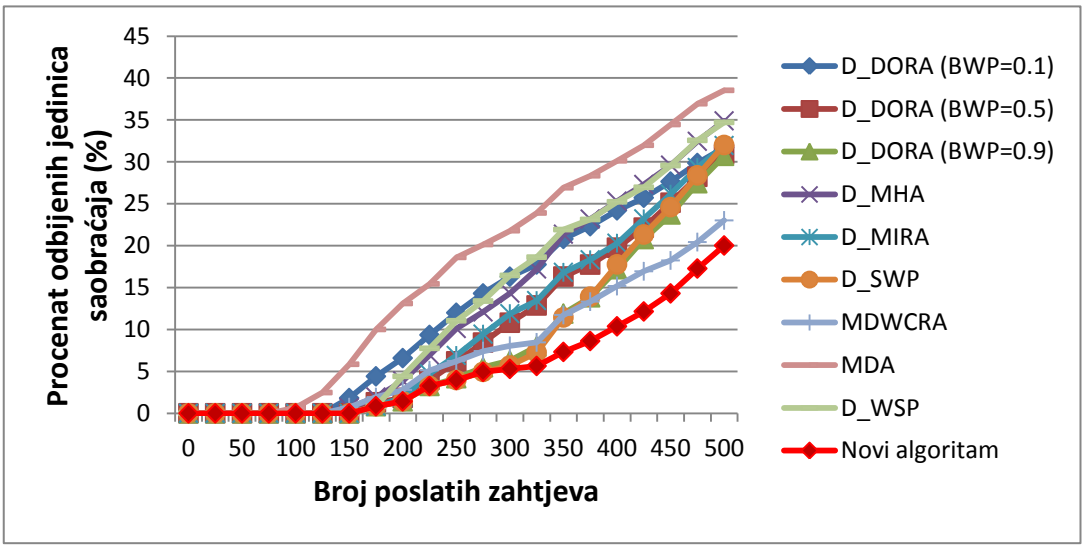
Slika 22: Procenat odbijenih jedinica DS saobraćaja, Dmax=110ms



Slika 23: Procenat odbijenih jedinica NDS saobraćaja, Dmax=110ms

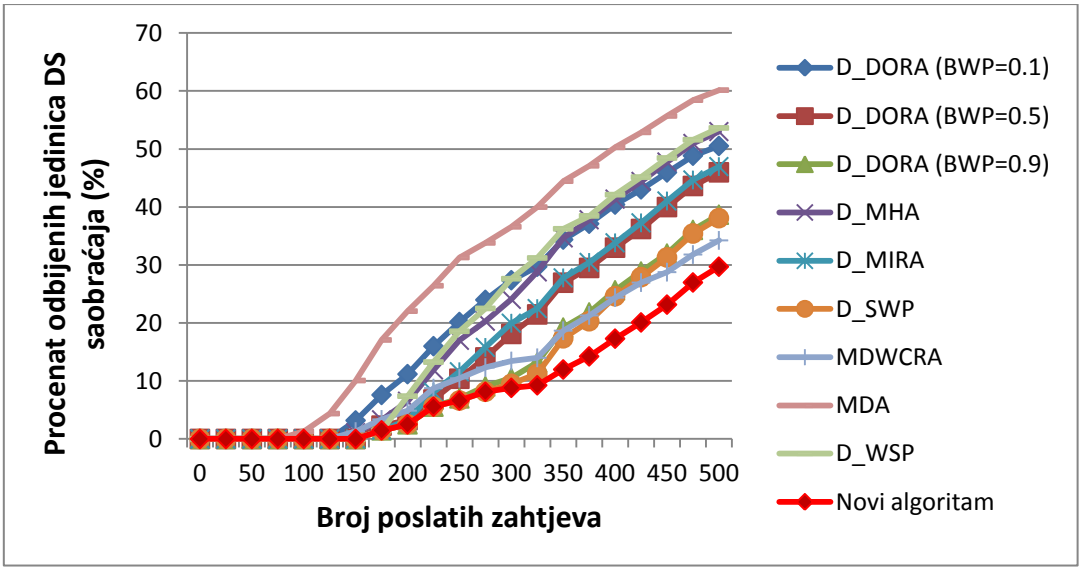
Performanse D_DORA algoritma u mnogome zavise od *BWP* parametra. Simulacije su izvršene za tri različite vrijednosti ovog parametra: 0.1, 0.5 i 0.9. Najbolji rezultati dobijeni su u poslednjem slučaju, koji akcenat stavlja na balansiranje opterećenja. Velikim vrijednostima *BWP* paramtera D_DORA algoritam ublažava diskutovane nedostatke D_MIRA algoritma, koji je takođe baziran na konceptu minimizacije interferencije između ulazno-izlazih čvorova. Sa druge strane, primjetno je da se u tom slučaju njegovo ponašanje približava D_SWP algoritmu, što potvrđuje da definisanje kritičnih linkova samo na osnovu slobodnog propusnog opsega ne doprinosi značajnijem poboljšanju performansi kada je kašenjenje jedno od ograničenja.

MDWCRA i algoritam koji je predložen u ovom magistarskom radu ističu se po performansama. Predstavljeni rezultati odgovaraju slučaju kada je težinski faktor MDWCRA algoritma obrnuto proporcionalan ukupnom kašnjenju kritičnih putanja. Za druga dva oblika tezinskog faktora (Sekcija 4.5) dobijeni su nešto lošiji rezultati. Poboljšanja u odnosu na D_DORA i D_MIRA algoritam, od koji je preuzeta ideja minimizacije interferencije između čvorova u mreži, ostaverena su drugačijim izborom kritičnih linkova. MDWCRA štiti *bottleneck* linkove nepreklapajućih putanja sa najmanjim kašnjenjem, i dodjeljuje im težinske faktore proporcionalno njihovom nivou kritičnosti za različite parove ulazno-izlaznih čvorova. Na ovaj način efikasno se održava potencijal za prihvatanje DS zahtjeva. U predloženom algoritmu ova ideja je još više prilagođena razmatranom scenariju sa različitim klasama saobraćaja, što je rezultovalo daljim poboljšanjima performansi. Strogim ograničavanjem samo na potencijalne DS putanje prilikom određivanja kritičnih linkova izbjegnuto je nepotrebno korišćenje dugačkih ruta. Sa druge strane, kritičnost linkova na DS putanjama je dodatno naglašena i ima snažniji uticaj na izbor ruta što je mreža zagušenija. Veća vjerovatnoća blokiranja NDS zahtjeva ne može se tumačiti kao loš rezultat, jer je prihvatanje DS zahtjeva od primarnog značaja. Ostali algoritmi odbili su manji broj jedinica NDS saobraćaja samo iz razloga što su prethodno blokirali veliki broj DS zahtjeva koji su MDWCRA i predloženi algoritam prihvatili.

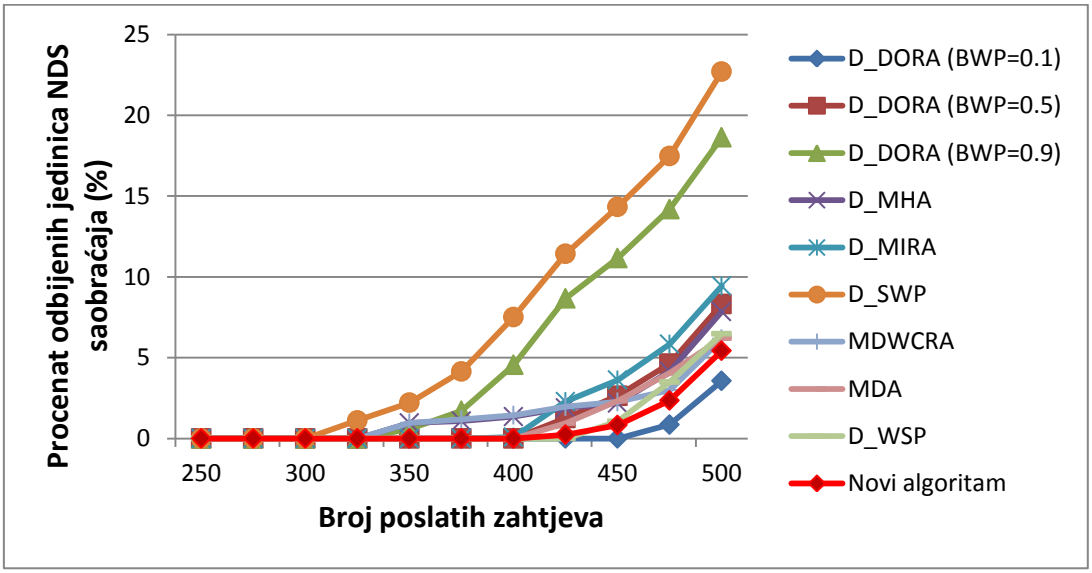


Slika 24: Procenat odbijenih jedinica saobraćaja, Dmax=90ms

Da bi se izloženi zaključci mogli generalizovati, simulacije su izvršene za različite vrijednosti maksimalnog dozvoljenog kašnjenja DS saobraćaja (90ms,100ms,110ms,120ms). U svakom od analiziranih slučajeva rangiranje algoritama po performansama je približno isto. MDWCRA i predloženi algoritam daju uvijek najbolje rezultate, a njihovo međusobno rangiranje ostaje nepromijenjeno. Radi ilustracije, na Slikama 24, 25 i 26 prikazani su rezultati za slučaj kada su ograničenja u pogledu kašnjenja strožija (Dmax=90ms).



Slika 25: Procenat odbijenih jedinica DS saobraćaja, Dmax=90ms



Slika 26: Procenat odbijenih jedinica NDS saobraćaja, Dmax=90ms

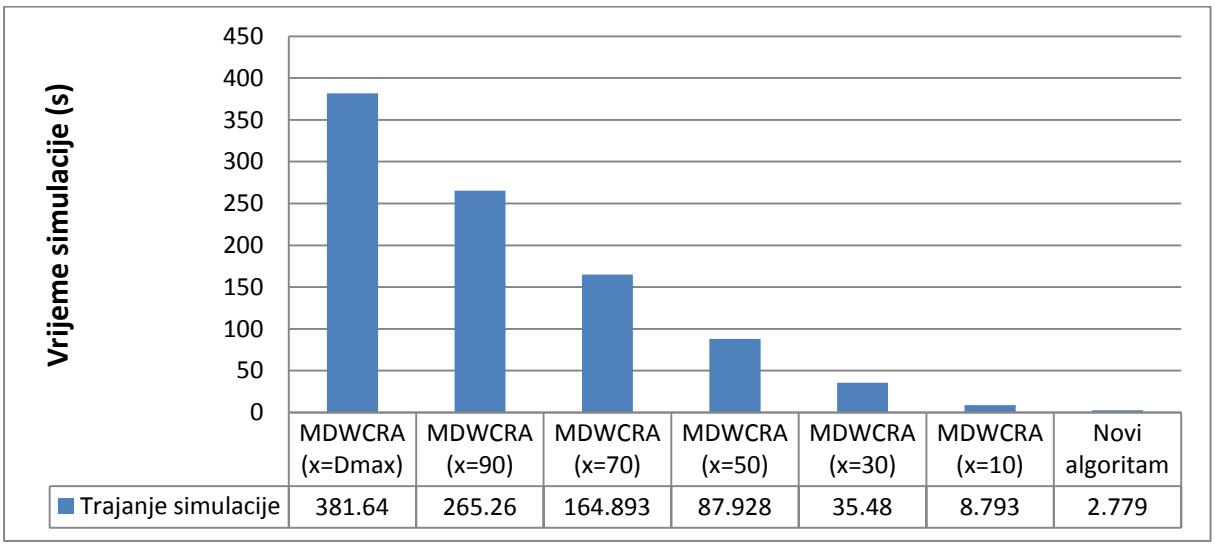
4.6.2 BRZINA IZVRŠAVANJA SIMULACIJE

Iako predstavljeni rezultati pokazuju da predloženi algoritam efikasnije koristi mrežne resurse, ključni nedostatak MDWCRA algoritma koji je motivisao predlog novog rešenja je prije svega njegova kompleksnost, odnoso veliko vrijeme izvršavanja. Slično kao DORA

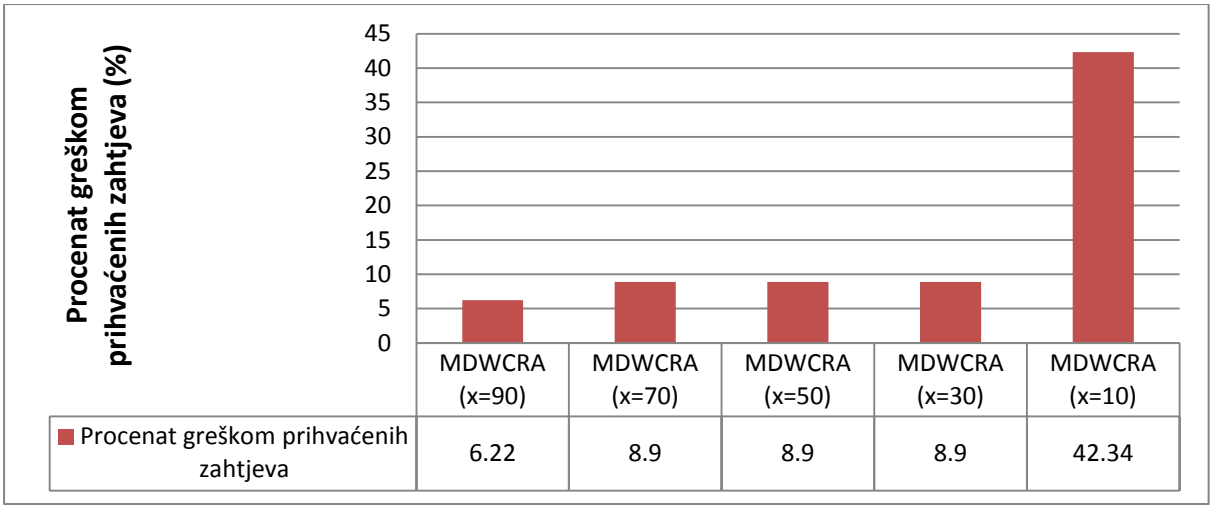
algoritam, MDWCRA određuje setove najkraćih nepreklapajućih putanja između parova ulazno-izlaznih čvorova. Složenost ove operacije je već diskutovana i sa primjenom Dijkstre iznosi $O(pNL)$, gdje je p broj parova ulazno-izlaznih čvorova u mreži. Računanje kritičnih linkova ne unosi dodatnu kompleksnost jer se može obavljati paralelno sa prethodnom operacijom. Glavni nosilac kompleksnosti je EDSP algoritam, čije je vrijeme izvršavanja $O(x^2N^2)$. Stoga, može se reći da je ukupna kompleksnost MDWCRA algoritma $O(pNL+x^2N^2)$. Sa druge strane u *online* fazi predloženog algoritma, u cilju pronalaženja kritičnih linkova, provjerava se svaka od K putanja. Najduža od K putanja je u najgorem slučaju dužine L, a svaka naredna je barem za jedan link kraća. Stoga, provjeravanje svakog od linkova zahtjeva $L+(L-1)+(L-2)+...+(L-K+1) \approx K*(L-K/2)$ operacija. Računski složena je jedino *offline* faza algoritma, u kojoj se vrši određivanje ovih putanja. Njeno vrijeme izvršavanja je $O(pkN^3)$, ali se ne obavlja u trenutku dolaska zahtjeva.

Ubrzavanje MDWCRA algoritma postiže se smanjivanjem parametra x, tj. izmjenom metrike kašnjenja prema formuli (11). Međutim, ovaj postupak unosi grešku koja je proporcionalna dužini rute. Kao rezultat može doći do uspostavljanja rute koja ne može da zadovolji zahtjeve konekcije. U cilju demonstracije ovog problema, u drugom eksperimentu je broj poslatih zahtjeva povećan na 1000 i mjereno je vrijeme trajanja simulacije i procenat greškom prihvaćenih zahtjeva. Maksimalno dozvoljeno kašnjenje za DS saobraćaj podešeno je na 110ms. Simulacije su pokrenute na PC računaru sa Intel i7 procesorom.

Na Slici 27 upoređene su vrijednosti trajanja simulacije predloženog i MDWCRA algoritma za različite vrijednosti x parametra. Može se primijetiti da MDWCRA znatno sporije obrađuje zahtjeve u svakoj od analiziranih situacija. Takođe, male vrijednosti x parametra značajno su se odrazile na korektnost algoritma. Pošto su u simulacijama zadate cjelobrojne vrijednosti kašnjenja, MDWCRA ne unosi grešku u slučaju kada je $x=Dmax$. Međutim, kao što se može vidjeti sa Slike 28, već za $x=90$ procenat greškom prihvaćenih zahtjeva iznosi 6.22 %, a pri vrijednosti $x=10$ čak 42.34 %. Bitno je napomenuti da je u simulacijama samo 60% zahtjeva bilo DS tipa, tako da se pri većem njihovom udjelu mogu očekivati još lošiji rezultati.

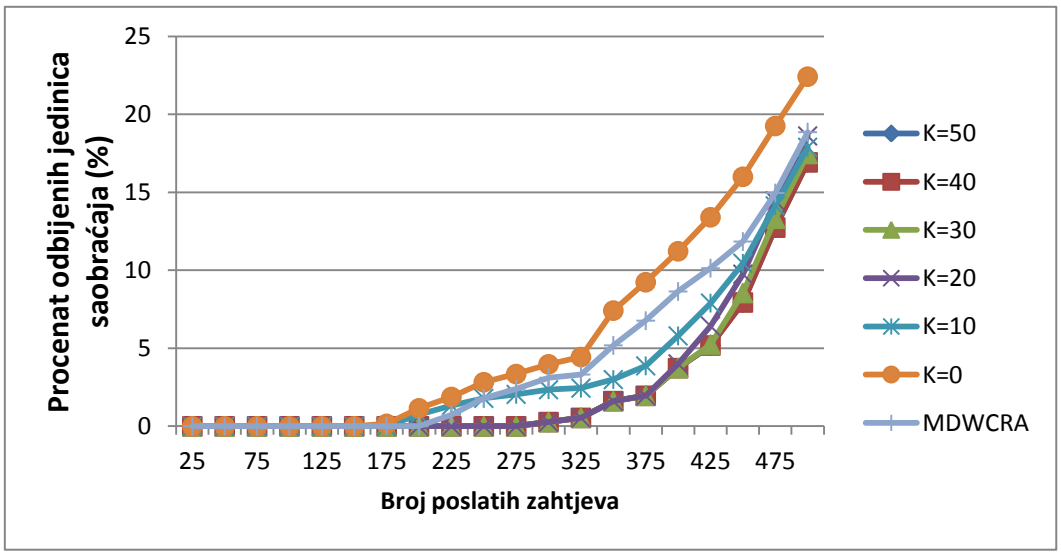


Slika 27: Trajanje simulacije u slučaju MDWCRA i predloženog algoritma



Slika 28: Procenat greškom prihvaćenih zahtjeva MDWCRA algoritma za različite vrijednosti *x* parametra

Brzina izvršavanja predloženog algoritma obrnuto je proporcionalna broju izračunatih DS putanja za svaki od parova. Set izračunatih putanja predstavlja uniju rešenja Yen-ovog algoritma i seta nepreklapajućih DS putanja posmatranog para. U prethodnim simulacijama algoritam je u *offline* fazi računao sve putanje između ulazno-izvornih čvrova koje odgovaraju zahtjevima DS saobraćaja. U zavisnosti od distribucije kašnjenja na linkovima ovaj broj varira, a samim tim i kompleksnost algoritma. Iz tog razloga algoritam nudi opciju za ograničavanje kompleksnosti definisanjem odgovarajuće vrijednosti *K* parametra. Na Slici 29 prikazani su rezultati analize uticaja ovog parametra na procenat odbijenih jedinica saobraćaja. Pri vrijednosti *K=50* u *offline* fazi uzete su u obzir sve moguće putanje između parova koje su prihvatljive za DS saobraćaj. Sa smanjivanjem K parametra performanse opadaju, ali i pri znatno manjim vrijednostima od "optimalne" ostaju bolje od performansi MDWCRA algoritma. Pri vrijednosti K=0 računate su samo nepreklapajuće DS putanje u *offline* fazi, i primjetne su znatno lošije performanse u tom slučaju.



Slika 29: Procenat odbijenih jedinica saobraćaja u funkciji od parametra *K*

4.6.3 PERFORMANSE U SCENARIJU SA DVIJE DS KLASE

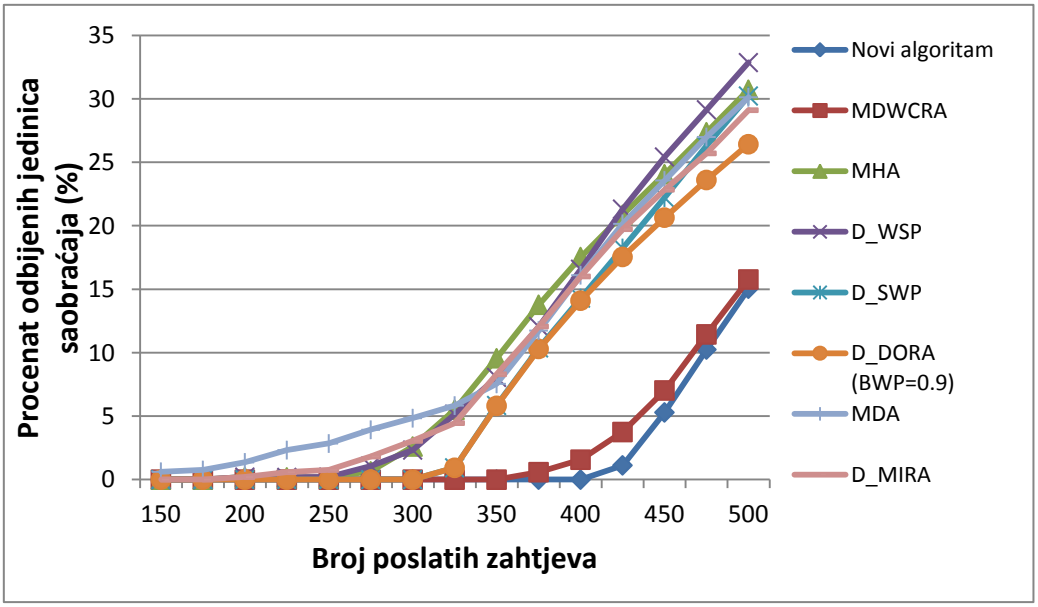
Ograničenje koje nameće predloženi algoritam je da zahtjevi u pogledu kašnjenja uzimaju jednu iz ograničenog seta vrijednosti. Takav pristup eliminiše potrebu za kompleksnim algoritmima kao što je EDSP, a predstavlja prihvatljivo rešenje za razmatrani scenario *backbone* mreže u kojima se kontrola vrši na nivou agregiranih saobraćajnih tokova. Podjela saobraćaja na DS i NDS klase predstavlja najgrublji oblik klasifikacije. Razlike u kašnjenjima koje tolerišu različiti tokovi ponekad mogu biti značajne, pa je poželjno omogućiti veći nivo granularnosti kada je klasifikacija u pitanju.

U ovoj sekciji analiziran je scenario sa dvije potklase DS saobraća: DS1 - koja zahtjeva kašnjenje manje od 90ms, i DS2 - koja toleriše kašnjenja manja od 120ms. Simulacija je podešana kao u sekciji 4.6.1, s tim što je 20% generisanog saobraćaja mapirano u DS1, a 40% u DS2 kategoriju. Ostatak kao i ranije čini NDS saobraćaj. Algoritam je modifikovan tako da u *offline* fazi za svaki od parova računa dva seta ruta koja odgovaraju različitim DS podklasama. U *online* fazi težinski faktor linka se povećava svaki put kada se link proglasi kritičnim za neku od putanja prema formuli:

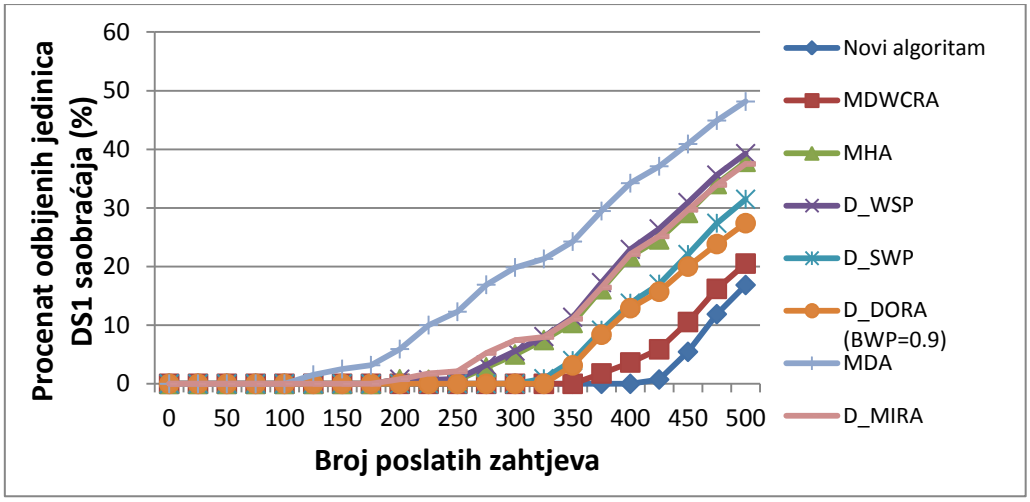
$$w(l)+ = \frac{1}{slobodni\_kapacitet(l) \cdot D_{\max}} \tag{14}$$

gdje *Dmax* predstavlja maksimalno dozvoljeno kašnjenje posmatrane DS potklase. Ukoliko je pristigli zahtjev DS1 tipa, kritični linkovi njegovog izvora i destinacije ne utiču na formiranje težinskog faktora. U slučaju DS2 zahtjeva samo kritični linkovi DS1 putanja odgovorajućeg para utiču na povećanje težinskih faktora.

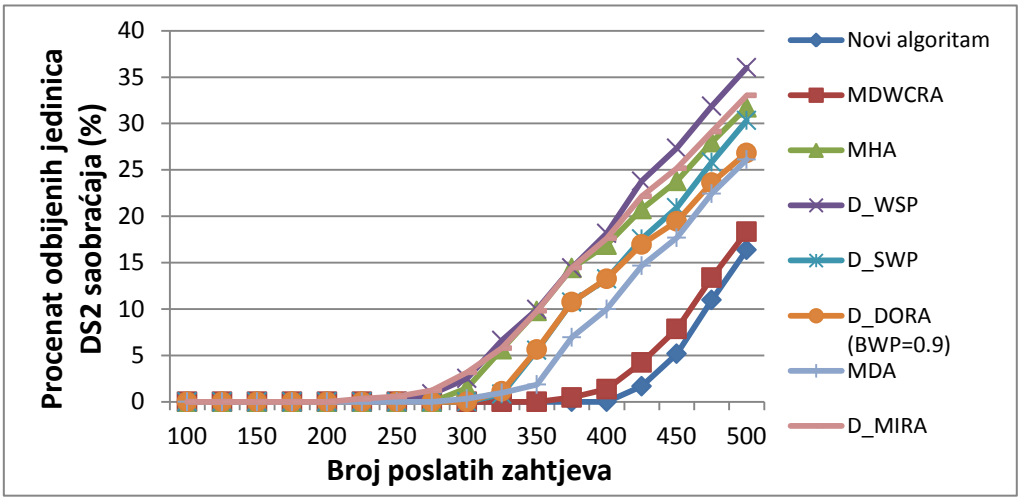
Rezultati u pogledu procenta odbijenih jedinica saobraćaja prikazani su na Slikama 30, 31, 32 i 33.



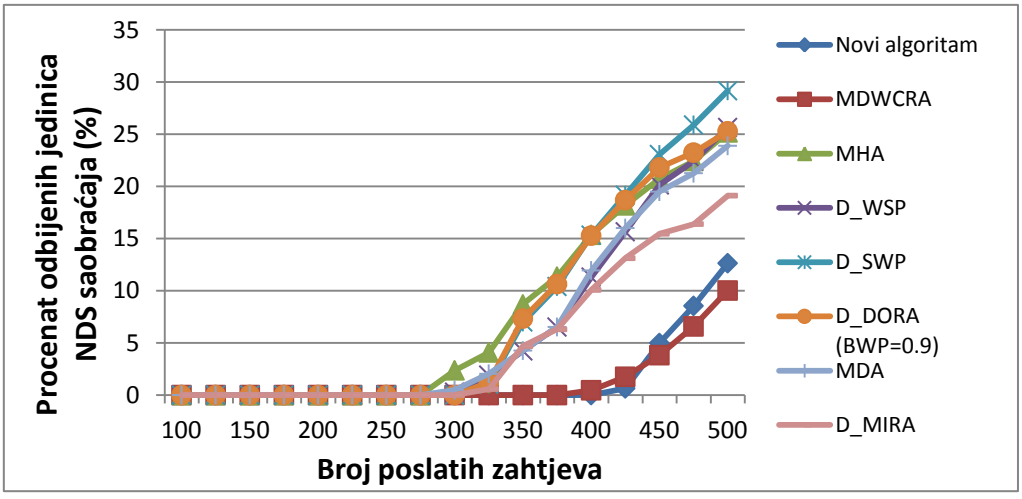
Slika 30: Procenat odbijenih jedinica saobraćaja u scenariju sa dvije DS potklase



Slika 31: Procenat odbijenih jedinica DS1 saobraćaja u scenariju sa dvije DS potklase



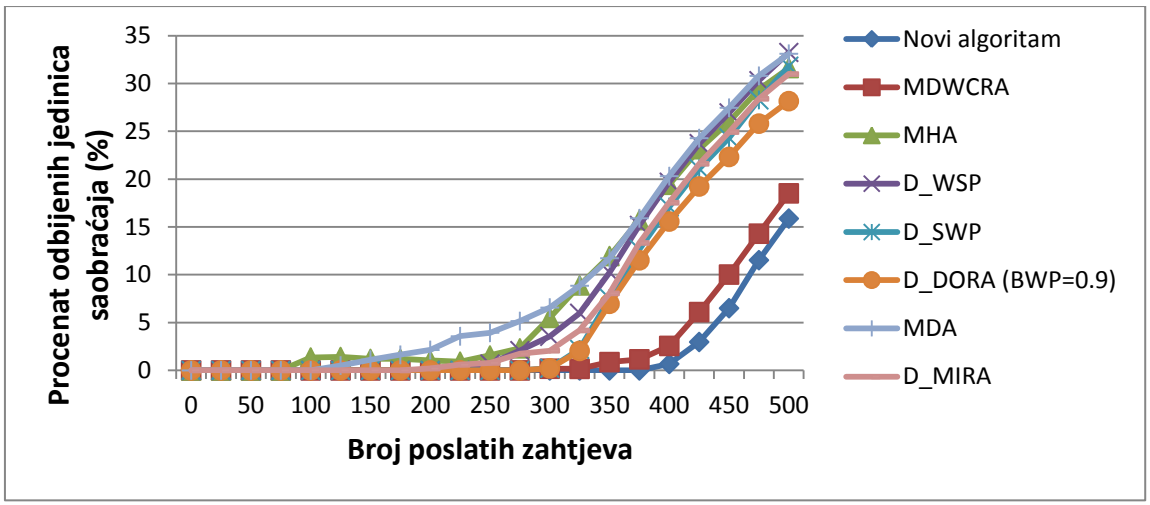
Slika 32: Procenat odbijenih jedinica DS2 saobraćaja u scenariju sa dvije DS potklase



Slika 33: Procenat odbijenih jedinica NDS saobraćaja u scenariju sa dvije DS potklase

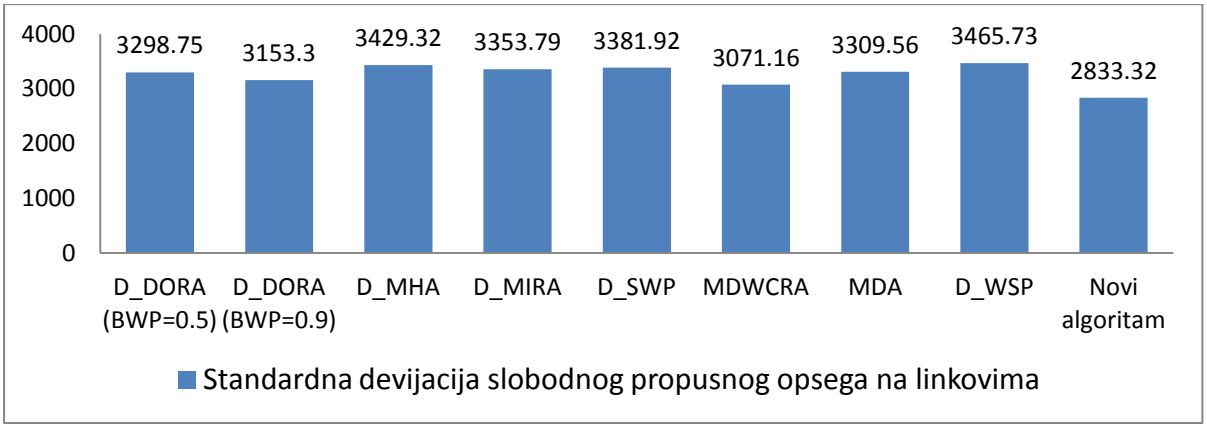
U ovom scenariju još je evidentnija razlika u performansama MDWCRA i predloženog algoritma na jednoj strani i ostalih analiziranih algoritama na drugoj strani.

D_DORA se i dalje rangira kao treća po performansama, ali neznatno odstupa od D_MHA, D_SWP, D_WSP i D_MIRA algoritama. Sa druge strane razlika u odnosu na MDWCRA i predloženi algoritam sada iznosi više od 20%. Prednost predloženog algoritma u odnosu na MDWCRA algoritam smanjena je usled većeg broja odbijenih NDS zahtjeva. Međutim, u praksi realno je očekivati da prihvatanje konekcija sa strogim zahtjevima u pogledu kašnjenja ima veću težinu od prihvatanja konekcija kojima su potrebne samo garancije u pogledu propusnosti. Simulacije su izvršene i za slučaj kada je NDS saobraćaj dominantno zastupljen. Predloženi algoritam i tada pokazuje najbolje performanse, iako štiti samo linkove koji pripadaju DS rutama (Slika 34).



Slika 34: Procenat odbijenih jedinica saobraćaja u slučaju kada je NDS saobraćaj dominantno zastupljen

Rezultati predstavljeni na Slici 35 pokazuju da predloženi algoritam najefikasnije balansira opterećenje linkova. Standardna devijacija slobodnog propusnog opsega važan je indikator sposobnosti algoritma da ravnomjerno raspoređuje konekcije u mreži. Što je ovaj parametar manji, za mrežu se može reći da je otpornija na otkaze linkova jer je veća vjerovatnoća pronalaženja alternativne rute.



Slika 35: Standardna devijacija slobodnog propusnog opsega na linkovima za različite algoritme

5. SDN KONTROLNO OKRUŽENJE ZA GARANCIJU KVALITETA SERVISA

SDN predstavlja tehnologiju u razvoju, pa danas dostupna rešenja SDN kontrolera ne nude mogućnost inteligentnog upravljanja saobraćajem. U vrijeme pisanja ove magistarske teze, funkcionalnost sakupljanja statističkih podataka iz mreže još uvijek nije integrisana na mnogim OpenFlow kontrolerima i primjeri efikasnih, skalabilnih i inteligentnih kontrolnih strategija nisu dostupne. U idealnom slučaju, kontroler može da prikuplja saobraćajne statistike od OpenFlow s*witch*-eva i koristi iste prilikom donošenja odluka o rutiranju. Ova mogućnost iskorišćena je za razvoj SDN okruženja za garanciju kvaliteta servisa na kojem se prethodno diskutovani algoritmi mogu implementirati.

5.1 QoS KONTROLNI SISTEM

Glavni cilj predloženog sistema je omogućavanje diferencijacije servisa i efikasnog korišćenja mrežnih resursa automatizovanom, visoko-granularnom kontrolom. Tradicionalna mrežna infrastruktura nije pogodna za ovu svrhu zbog svoje statičke i kompleksne prirode. Ovo je posledica distribuirane kontrole ravni, koja čini mrežu teškom za konfigurisanje. Iako postojeće mreže mogu da pruže različit nivo kvaliteta servisa različitim aplikacijama, rezervacija resursa zahtjeva dosta manuelne intervencije. Na primjer, administrator mora konfigurisati veliki broj uređaja pojedinačno kako bi podesio QoS parametre na nivou sesije ili aplikacije. Posledično, mreža nije u stanju da se dinamički prilagođava promjenjivim zahtjevima korisnika i aplikacija. U cilju automatizacije konfigurisanja i upravljanja, predloženi sistem baziran je SDN/OpenFlow arhitekturi, koja uz odgovarajuće programe na kontroleru može značajno doprinijeti optimizaciji mrežnih resursa [78][79].

Dostupna rešenja *open-source* kontrolera još uvijek nemaju implementiranu podršku za garanciju kvaliteta servisa. Kao moguće rešenje, u ovom magistarskom radu predlaže se dizajn QoS SDN/OpenFlow kontrolera sa Slike 36. Ključni funkcionalni blokovi koje dizajn uključuje su:

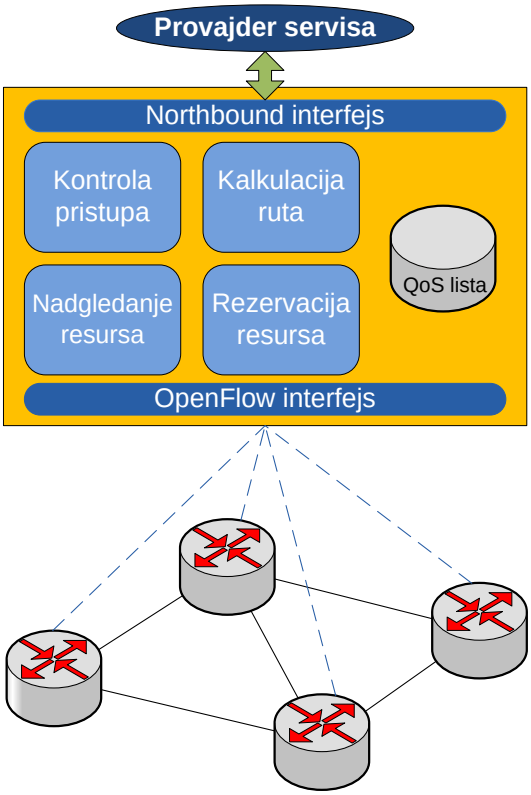
- nadgledanje resursa,
- kalkulacija ruta,
- kontrola pristupa (CAC - *Call Admission Control*) i
- rezervacija resursa.

U nastavku poglavlja biće diskutovana implementacija svake od navedenih funkcionalnosti pojedinačmo.

5.1.1 MODUL ZA NADGLEDANJE RESURSA

Modul za nadgledanje resursa odgovoran je za prikupljanje i održavanje informacije o trenutnom stanju u mreži. Prije svega, fundamentalni zadatak ovog modula je detekcija

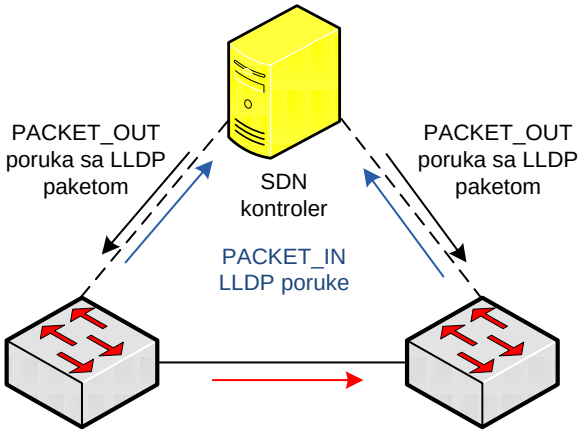
topologije, što podrazumijeva detekciju: 1) OpenFlow *switch*-eva, 2) linkova, i 3) hostova u mreži. Kada OpenFlow *switch* uspostavi TCP konekciju sa kontrolerom, kontroler šalje *FEATURE_REQUEST* poruku *switch*-u i čeka odgovor. Kada stigne poruka odgovora, kontroler izvlači informacije o karakteristikama *switch*-a, kao što su na primjer identifikator *switch*-a (*datapath ID*), lista portova itd. Mrežni uređaji koriste blagu modifikaciju LLDP (*Link Layer Discovery Protocol*) protokola za oglašavanje i identifikaciju susjeda. Ovi LLDP frejmovi se prepoznaju u mreži na osnovu specijalne MAC adrese i *Ethernet Type* polja podešenog na vrijednost 0x08cc. Proces detekcije topologije inicira kontroler koji enkapsulira LLDP pakete u *PACKET_OUT* poruke i periodično ih šalje svim povezanim uređajima. Poruke upućuju uređaje da dalje šalju LLDP pakete preko svih svojih interfejsa. Kada susjed primi paket ovog tipa, vrši pretraživanje tabele tokova. Kako za specijalne LLDP pakete ne postoji odgovarajući zapis, paket se šalje kontroleru putem *PACKET_IN* poruke koja je definisana OpenFlow specifikacijom. Analizirajući *PACKET_IN* poruke kontroler identifikuje na koji način su fizički uređaji povezani u mreži (Slika 37). Ove informacije su konstantno dostupne ostalim modulima koji se izvršavaju na kontroleru. U slučaju ispada ili dodavanja novog linka modul oglašava događaj da obavijesti registrovane osluškivače.



Slika 36. Predloženi dizajn QoS SDN arhitekture [78][79]

Detekcija lokacije hostova vrši se reaktivno. Ukoliko host prethodno nije generisao saobraćaj u mrežu, OpenFlow *switch*-evi neće imati instrukcije za obradu njegovih paketa pa će ih slati kontroleru. Kontroler kada prvi put opazi paket od novog hosta memoriše identifikator OpenFlow *switch*-a i broj porta na koji je paket stigao. Međutim, ukoliko nije poznata lokacija destinacione adrese kontroler nije u stanju da donese odluku o rutiranju. U tom slučaju šalje se instrukcija za *broadcast*-ovanje paketa. Petlje se izbjegavaju primjenom

funkcije koja oponaša STP (*Spanning Tree Protocol*) protokol [80]. Bitna razlika u odnosu na STP je da se kreirano stablo koristi samo za prosleđivanje *broadcast* saobraćaja. Za prenos *unicast* saobraćaja i dalje se mogu korisiti svi interfejsi uređaja. Tek kada destinacioni host bude generisao saobraćaj kontroler će saznati lokaciju čvorova između kojih je potrebno uspostaviti rutu.



Slika 37: Proces detekcije topologije u OpenFlow mreži

Pored funkcije detekcije topologije, koja je već implementirana na dostupnim OpenFlow kontrolerima, modul za nadgledanje resursa je nadograđen sa funkcijom za prikupljanje statističkih podataka. Kontroler je isprogramiran da periodično šalje specijalne OpenFlow poruke zahtjeva za različite vrste statistika. Brojači implementirani na OpenFlow *switch*-evima omogućavaju uvid u broj bajtova poslatih preko svakog interfejsa ili saobraćajnog toka. Ovi "sirovi" podaci koriste se za dobijanje korisnih informacija, kao što su propusnost toka i opterećenje linka, koje se kasnije stavljaju na raspolaganje modulu za kalkulacije ruta. Takođe, kontroler održava informacije o rezervisanom propusnom opsegu za svaki od QoS tokova.

Obrada statističkih podataka je veoma zahtjevna jer je potrebno računati propusnost za potencijalno veliki broj tokova i linkova. Stoga, veoma je važno rasteretiti kontroler od intenzivnog računanja što je više moguće. Iz ovog razloga samo pristupni OpenFlow *switch* toka se provjerava za potrebne statistike. U cilju redukcije kontrolnog soabraćaja kontroler šalje samo jedan zahtjev za statistike svih tokova instaliranih na *switch*-u. Odgovor je takođe enkapsuliran u jednoj poruci koja sadrži status brojača za svaki od tokova. Nepotrebne informacije se jednostavno odbacuju. Interval između uzastopnih zahtjeva podešen je na 3 sekunde. Može se reći da je na ovaj način napravljen kompromis, s obzirom da ne samo prevelike, već i premale vrijednosti ovog intervala negativno utiču na tačnost dobijenih informacija. Open vSwitch [81], koji je korišćen u realizaciji *testbed*-a, ažurira vrijednosti brojača u korisničkom prostoru sa vrijednostima iz kernela u intervalima od jedne sekunde [82], tako da češće obavljanje mjerenja samo nepotrebno zagušuje kontroler.

5.1.2 MODUL ZA KALKULACIJU RUTA

Ovaj modul određuje rute za različite tipove saobraćaja. Nekoliko algoritama rutiranja

može se koristiti paralelno kako bi se zadovoljili različiti zahtjevi. Kao ulazne argumente uzima rezultate modula za nadgledanje resursa i listu tokova koji zahtjevaju QoS garancije.

Prilikom donošenja odluka o rutiranju QoS saobraćaja kontroler konstruiše redukovani graf koji sadrži samo one linkove koji imaju dovoljno slobodnog propusnog opsega da ispune zahtjev u pogledu propusnosti. Međutim, pod slobodnim propusnim opsegom podrazumijeva dio propusnog opsega koji nije već rezervisan za QoS tokove. Ovo može dovesti do značajne degradacije performansi *best-effort* saobraćaja ukoliko se on rutira na standardni način - najkraćom putanjom. Imajući u vidu da je *best-effort* saobraćaj dominantni tip saobraćaja na Internetu, čije se performanse ne mogu zanemariti, umjesto broja hopova metrika rutiranja definisana je kao funkcija estimiranog nivoa zagušenja linka. Računanje rutra se obavlja Dijkstra algoritmom, ali težinski faktori određuju se prema formuli:

$$w(l) = \frac{C(l)}{C(l) - \max(rez(l), est(l))} \tag{15}$$

gdje *C(l)* označava kapacitet linka *l*, a *rez*(*l*) i *est*(*l*) rezervisani i zazuzeti propusni opseg na linku redom. Vrijednost *est(l)* zapravo predstavlja rezultat modula za nadgledanje resursa, i jednak je estimiranom opterećenju linka u poslednjem ciklusu mjerenja. Na ovaj način algoritam izbjegava zagušene linkove, čak i ukoliko je saobraćaj na njima *best-effort*. Težinski faktor rapidno raste sa stepenom iskorišćenja linka, što omogućava bolje performanse kada je mreža teško opterećena. Funkcija maksimuma u imeniocu koristi se da kompenzuje potencijalne greške rezultata dobijenih mjerenjem.

Definisanje težinskog faktora na način koji je prethodno opisan i dalje ne sprečava degradaciju postojećeg *best-effort* saobraćaja prilikom uspostavljanja novih QoS tokova. Iz ovog razloga kontroler je programiran da po potrebi preventivno rerutira *best-effort* saobraćaj kako bi se izbjeglo zagušenje. Nakon što se izračuna ruta za QoS tok, za svaki link rute kontroler provjerava da li će doći do prekoračenja određenog praga iskorišćenosti (80% u implementiranom modelu). Ukoliko je to slučaj, kontroler računa količinu propusnog opsega koju je potrebno osloboditi tako da se iskorišćenost linka svede ispod nivoa praga. Ova informacija koristi se kao ulazni argument algoritma rerutiranja, čiji je pseudo-kod dat Algoritmom 4. U osnovi, algoritam je modifikacija rešenja iz literature [83]. Ključna ideja je da se oslobodi potrebna količina propusnog opsega rerutiranjem što manjeg broja *best-effort* tokova. Iako ovo nameće značajnu računsku kompleksnost, modifikacije tabela tokova zahtjevaju znatno više vremena neko ostale operacije, pa je potrebne izmjene poželjno svesti na minimum.

Prvo, algoritam provjerava da li je potrebni propusni opseg moguće osloboditi rerutiranjem samo jednog toka (linija 8). Međutim, iako se pronađe takav tok, biće rerutiran samo ukoliko postoji putanja za koju se procjenjuje da ne može dovesti u stanje zagušenja. Ovo se provjerava *probaj_rerutiranje()* funkcijom koja kao rezultat vraća količinu oslobođenog propusnog opsega ukoliko je tok uspješno rerutiran, ili nulu u suprotnom. Pseudo-kod funkcije dat je Algoritmom 5. Treba primijetiti da prije provjere mogućnosti migracije informacije o iskorišćenosti linkova moraju biti ažurirane tako da ne uzimaju u

obzir saobraćajno opterećenje ispitivanog toka. Ukoliko pokušaj oslobađanja potrebnog propusnog opsega propadne, algoritam pokušava da ostvari isti cilj sa višestrukim tokovima. (linije 16-20). Ispitivanje ponovo počinje od najvećeg neispitanog toka kako bi se rerutiranje minimizovalo.

Algoritam 4: Rerutiranje best-effort saobraćaja

```
1:#tok.p_opseg - propusni opseg toka
2:#be_tokovi - lista best-effort tokova na određenom linku
3:#potrebni_opseg - količina propusnog opsega koju treba osloboditi
4:#len(be_tokovi) - broj elemenata liste be_tokovi
5:
6: function:  rerutiraj_best_effort(be_tokovi, potrebni_opseg)
7:      be_tokovi.sortiraj()                          #sortranje u rastući poredak
8:      if be_tokovi[len(be_tokovi)-1].p_opseg ≥ potrebni_opseg then:
9:          for tok in be_tokovi do:
10:             if tok.p_opseg ≥ potrebni_opseg then:
11:                 potrebni_opseg -= probaj_rerutiranje(tok)
12:                 be_tokovi.ukloni(tok)
13:                 if potrebni_opseg ≤ 0 then:
14:                     break for-loop
15:
16:     while potrebni_opseg > 0 and be_tokovi ≠ Ø do:
17:                 tok = be_tokovi[len(be_tokovi)-1]
18:                 potrebni_opseg -= probaj_rerutiranje(tok)
19:                 be_tokovi.ukloni(flow)
20:     end while
21:end function
```

5.1.3 MODUL ZA REZERVACIJU RESURSA

Ovaj modul pruža *end-to-end* garancije propusnosti za QoS tokove tako što konfiguriše izlazne bafere mrežnih uređaja. *ENQUE* akcija definisana OpenFlow protokolom omogućava mapiranje paketa sa odgovarajućim izlaznim baferom. Međutim, iako kontroler može da šalje instrukcije ovog tipa, konfiguracija bafera je za sada izvan OpenFlow specifikacije. OF-Config protokol [84] je obećavajuće rešenje za automatizaciju konfigurisanja u mrežama baziranim na OpenFlow verziji 1.2 pa nadalje. U našem eksperimentalnom okruženju baziranom na OpenFlow verziji 1.0 ovo ograničenje je prevaziđeno tako što je kontroler isprogramiran da šalje konfiguracione komande. S obzirom da se *testbed* sastoji od softverskih OpenFlow *switch*-eva, ove komande zapravo koriste ugrađenu mogućnost Linux sistema za oblikovanje saobraćaja. Preciznije, opsluživanje saobraćaja vrši se HTB (*Hierarchical Token Bucket* [85]) algoritmom, koji omogućava garantovanje minimalne i ograničavanje maksimale brzine prenosa toka. Čim se kontroler poveže na mrežu modul za rezervaciju resursa kreira specijalan bafer za *best-effort* saobraćaj na svakom mrežnom interfejsu. Kada kontroler primi paket koji pripada nekom od QoS tokova, na svim odlaznim interfejsima izračunate putanje kreira se novi bafer i konfiguriše njegova minimalna odnosno maksimalna brzina prenosa. Preko OpenFlow protokola dodaje

se novi zapis u tabele tokova svih OpenFlow *switch*-eva na ruti, koji filtrira pakete tog QoS toka i šalje ih u kreirani bafer. *Best-effort* saobraćaju se ne garantuje minimalna brzina prenosa, ali se ograničenje u pogledu maksimalne brzine podešava na puni kapacitet linka.

Algoritam 5: Provjera mogućnosti rerutiranja

```
1:#L(G)  - set linkova mreže
2:#redukovana_topologija - set linkova koji mogu prihvatiti tok a da ne uđu u stanje zagušenja
3:#prag - prag zagušenja
4:
5:  redukovana_topologija=Ø
6:  function: probaj_rerutiranje(tok):
7:      for link in tok.ruta do:
8:          link.zauzetost-=tok.p_opseg
9:      for link in L(G)  do:
10:         if link.zauzetost < prag-tok.p_opseg then:
11:             redukovana_topologija.dodaj(link)
12:    nova_ruta=Dijkstra(G,tok.izv,tok.dest)
13:    if nova_ruta ≠ Null then:
14:        for link in nova_ruta do:
15             link.zauzetost+=tok.p_opseg
16:        rerutiraj(tok, nova ruta)
17:        return tok.p_opseg
18:    else:
19:        for link in tok.ruta do:
20:            link.zauzetost += tok.p_opseg
21:        return 0
22: end function
```

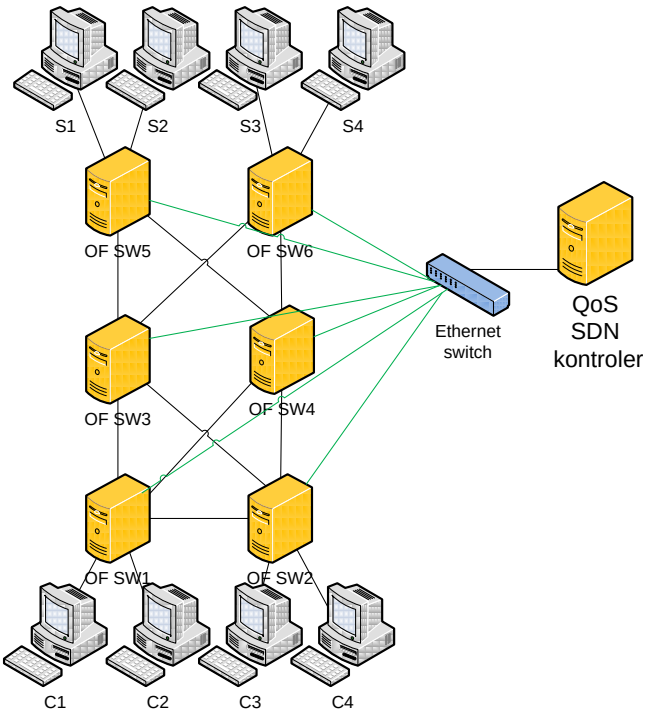
5.1.4 KONTROLA PRISTUPA

Modul za kontrolu prostupa sprovodi odbijanje QoS zahtjeva ukoliko se ustanovi da ne postoje uslovi za njihovo ispunjenje i šalje klijentu povratnu informaciju. Cilj eksperimenata čiji su rezultati predstavljeni u nastavku poglavlja bio je potvrda mogućnosti kontrole kvaliteta servisa primjernom SDN-a, tako da realizaciji interfejsa prema servis provajderu nije posvećena posebna pažnja. Saobraćaj je generisan u eksperimentalnim uslovima, pa su definicije QoS tokova čuvane u specijalnom fajlu čiji sadržaj SDN kontroler periodično provjerava. Na sličan način vodi se evidencija o blokiranim zahtjevima.

5.2 EKSPERIMENTALNI REZULTATI

Za implementaciju predloženog dizajna izabran je POX kontroler [56] koji je realizovan u Python-u. U eksperimentima čiji su rezultati prikazani u ovoj sekciji korišćeno je testno mrežno okruženje sa Slike 38. Ravan podataka sastoji se od šest Linux računara na kojima je pokrenut Open vSwitch softver. Računari su međupovezani tako da postoji više

nepreklapajućih putanja između svakog para klijent-server. Sve mrežne kartice su kapaciteta 100 Mb/s.



Slika 38: Testbed (C-klijent, S-server)

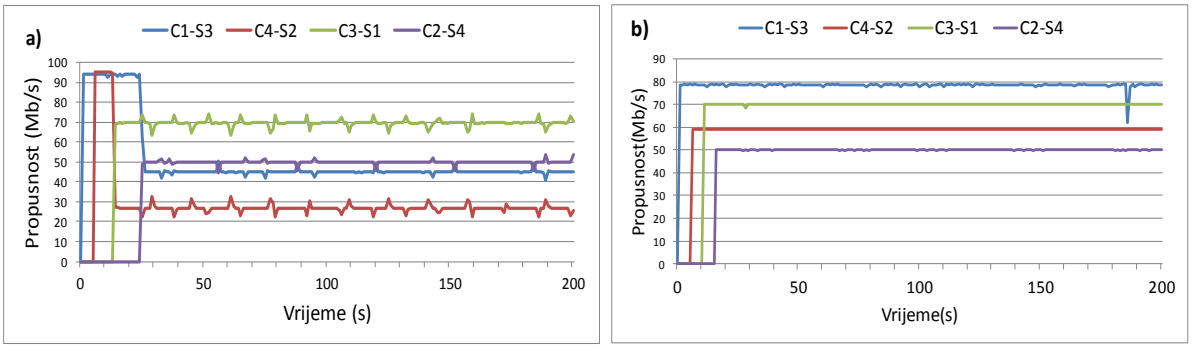
Jasno je da na mreži ovako malih razmjera nije moguće izvršiti poređenje QoS algoritama koji su analizirani u četvrtom poglavlju. S obzirom da implementacija *backbone* mreže nije realna, eksperimenti koji su napravljeni na *testbed*-u za cilj su imali prije svega provjeru funkcionalnosti razvijenog OpenFlow kontrolera sa podrškom za garanciju kvaliteta servisa. Predstavljeni rezultati dobijeni su u početnoj fazi istraživanja kada je na kontroleru bio implementiran MDA algoritam za rutiranje QoS saobraćaja. Imajući u vidu modularni dizajn kontrolera, implementacija bilo kojeg od analiziranih algoritama nije upitna, ali se ne odražava na dobijene rezultate zbog malog diverziteta putanja i izvora saobraćaja. Međutim, razvoj kontrolnog okruženja za garanciju kvaliteta servisa od velikog je značaja za potencijalna buduća istraživanja na *testbed*-ovima velikih istraživačkih kampusa.

Na implementiranom OpenFlow *testbed*-u sprovedena su tri različita eksperimenta u cilju poređenja performansi predložene SDN QoS arhitekture sa tradicionalnim *best-effort* i Intserv modelom servisa. *Best-effort* model servisa imitiran je pokretanjem *l2_multi* aplikacije, koja dolazi u sklopu POX kontrolera i vrši rutiranje najkraćom putanjom. Specijalna aplikacija je kreirana za imitaciju IntServ ponašanja. U osnovi, ova skripta samo upotpunjuje rutiranje najkraćom putanjom sa funkcionalnošću rezervacije resursa. S obzirom da IntServ rezerviše resurse u mreži putem RSVP protokola, koji nije protokol rutiranja, glavna mana ovog modela je što se QoS zahtjev odbija ukoliko nema dovoljno resursa na ruti sa najmanjim brojem hopova. Ova fundamentalna karakteristika IntServ-a reflektovana je u kreiranoj SDN aplikaciji.

U prvom eksperimentu generisana su 2 TCP i 2 UDP toka *iperf* softverskim alatom [86]. Njihove karakteristike opisane su u Tabeli 4. Na Slikama 39a i 39b prikazani su rezultati u pogledu propusnosti koji je ostvario svaki od njih sa *l2_multi* aplikacijom i predloženom QoS realizacijom kontrolera. Kada je sav saobraćaj tretiran kao *best-effort* došlo je do značajne degradacije propusnosti TCP tokova nakon uvođenja UDP saobraćaja. Usled odsustva mehanizama za kontrolu zagušenja kod UDP protokola, tokovi ovog tipa monopolizovali su propusni opseg i doveli TCP tokove u starvaciju. U ovom konkretnom slučaju, TCP tokovi dobili su samo dio propusnog opsega koji UDP tokovi nisu zahtijevali na istoj ruti. Sa druge strane, QoS kontroler je iskoristio informacije sakupljenje modulom za nadgledanje mreže i distribuirao tokove nepreklapajućim putanjama. Takođe, resursi su rezervisani za svaki od njih, pa uvođenje UDP tokova nije moglo da utiče na performanse TCP saobraćaja.

Tabela 4: Karakteristike tokova u eksperimentu 1:

Tok	***Izv.-Dest.***	***Transp. Prot.***	***Brzina prenosa***	***Tip***
1.	C1-S3	TCP	80Mb/s	QoS
2.	C4-S2	TCP	60Mb/s	QoS
3.	C3-S1	UDP	70Mb/s	QoS
4.	C2-S4	UDP	50Mb/s	QoS

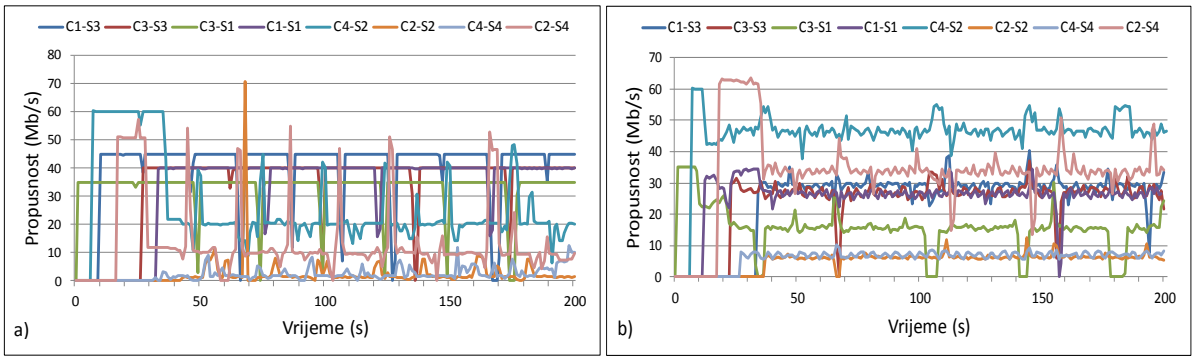


Slika 39: Eksperiment 1 - Ostvarena propusnost sa a) *l2_multi* aplikacijom i b) QoS SDN kontrolerom

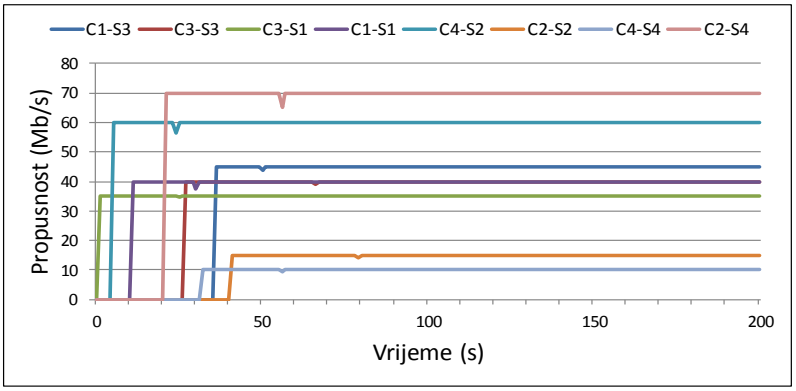
U drugom eksperimentu generisano je 8 UDP tokova opisanih u Tabeli 5. Ovoga puta performanse predloženog QoS rešena upoređene su i sa IntServ modelom. Na Slikama 40a i 40b prikazani su rezultati u pogledu ostvarene propusnosti za IntServ i *best-effort* scenario. U oba slučaja, tokovi su iskusili mnogo lošije performanse nego što je mreža bila u stanju da pruži. Ovo je posledica korišćenja uvijek istog stabla najkraćih putanja prilikom rutiranja. IntServ je uspio da prihvati samo četiri QoS zahtjeva (tokove 1, 3, 5 i 7 iz Tabele 4) jer su se stabla najkraćih putanja različih izvora preklapala. Međutim, može se primjetiti jaka povremena degradacija ovih tokova. Razlog je taj što je veliki broj tokova rutiran preko OpenFlow *switch*-a 4, koji je zbog svojih softverskih ograničenja povremeno imao problema da se izbori sa velikim saobraćajnim opterećenjem. Na Slici 50 prikazani su rezultati u pogledu propusnosti za slučaj kada je korišćena predložena arhitektura SDN kontrolera. Kontrolna aplikacija je ponovo iskoristila diverzitet ruta, i pružila zahtjevani kvalitet servisa svakom od tokova.

Tabela 5: Karakteristike tokova u eksperimentu 2

Tok	*Izv.-Dest.*	*Transp. Prot.*	*Brzina prenosa*	*Tip*
1.	C3-S1	UDP	35Mb/s	QoS
2.	C4-S2	UDP	60Mb/s	QoS
3.	C1-S1	UDP	40Mb/s	QoS
4.	C2-S4	UDP	70Mb/s	QoS
5.	C3-S3	UDP	40Mb/s	QoS
6.	C4-S4	UDP	10Mb/s	QoS
7.	C1-S3	UDP	45Mb/s	QoS
8.	C2-S2	UDP	15Mb/s	QoS



Slika 40: Eksperiment 2 - Ostvarena propusnost sa a) *IntServ* aplikacijom i b) *l2_multi* aplikacijom

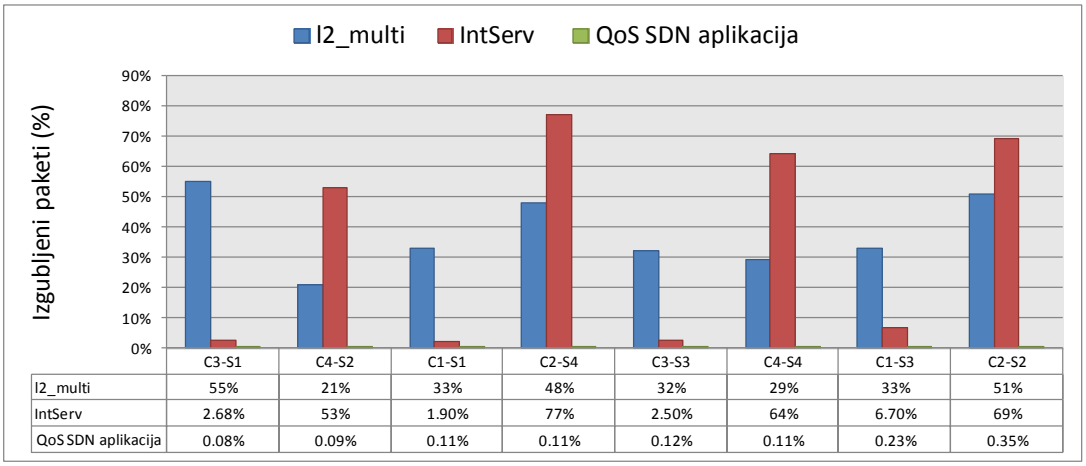


Slika 50: Eksperiment 2 - Ostvarena propusnost sa QoS SDN kontrolerom

Rezultati u pogledu procenta izgubljenih paketa u drugom eksperimentu prikazani su na Slici 51. Predloženo kontrolno rešenje jasno odstupa po performansama od *best-effort* i IntServ modela servisa. Kao što je očekivano, tokovi odbijeni IntServ aplikacijom izgubili su veći broj paketa u IntServ nego u *best-effort* scenariju jer nisu mogli da koriste resurse koji su rezervisani za prihvaćene tokove.

U trećem eksperimentu ispitan je uticaj predloženog kontrolnog rešenja i IntServ modela na performanse *best-effort* saobraćaja. Prvo je generisano 6 *best-effort* tokova jedan za drugim prema redosledu definisanim u Tabeli 6. Nakon toga pušten je QoS tok zahtjeva 45 Mb/s. U IntServ scenariju svi tokovi su rutirani istom rutom, što je rezultovalo značajnom

degradacijom performansi *best-effort* saobraćaja nakon uvođenja QoS toka (Slika 52a). QoS kontroler je ujednačeno distribuirao *best-effort* opterećenje preko OF1-OF3-OF5 i OF1-OF4-OF5 ruta (51Mb/s na svakoj). Ovo je ostvareno mehanizmima rutiranja koju su opisani u Sekciji 5.1.2. Kasnije, jedna od ovih rutra izabrana je za QoS tok. Međutim, kontroler je brzo identifikovao da će doći do prekorčenja praga zagušenja (80Mb/s) i pokrenuo algoritam rerutiranja. U skladu sa kontrolnom logikom opisanom Algoritmima 4 i 5, tok od 20Mb/s migriran je na alternativnu rutu. Ostvarena poboljšanja jasno se vide na Slici 52b.

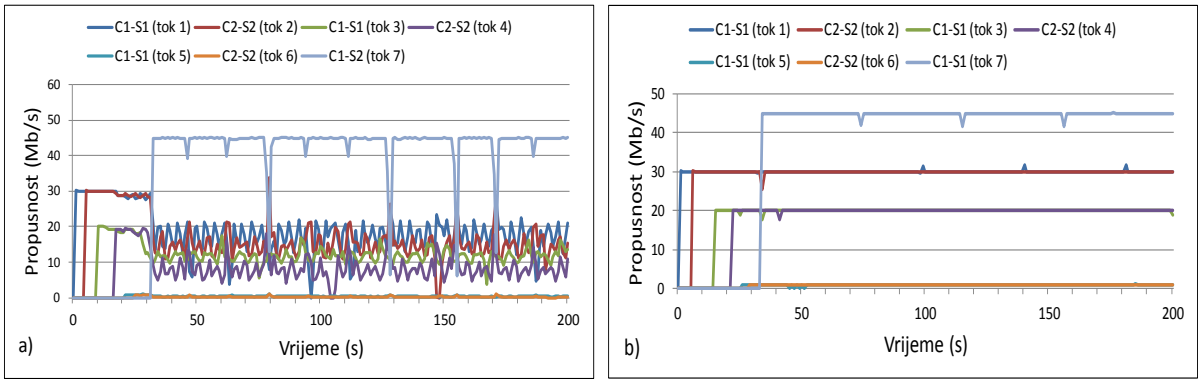


	C3-S1	C4-S2	C1-S1	C2-S4	C3-S3	C4-S4	C1-S3	C2-S2
l2_multi	55%	21%	33%	48%	32%	29%	33%	51%
IntServ	2.68%	53%	1.90%	77%	2.50%	64%	6.70%	69%
QoS SDN aplikacija	0.08%	0.09%	0.11%	0.11%	0.12%	0.11%	0.23%	0.35%

Slika 51: Eksperiment 2 - Procenat izgubljenih paketa

Tabela 6: Karakteristike tokova u trećem eksperimentu

Tok	***Src.- Dest.***	***Transp. Prot.***	***Brzina prenosa***	***Tip***
1.	C1-S1	UDP	30Mb/s	Best-effort
2.	C2-S2	UDP	30Mb/s	Best-effort
3.	C1-S1	UDP	20Mb/s	Best-effort
4.	C2-S2	UDP	20Mb/s	Best-effort
5.	C1-S1	UDP	1Mb/s	Best-effort
6.	C2-S2	UDP	1Mb/s	Best-effort
7.	C1-S1	UDP	45Mb/s	QoS



Slika 52: Eksperiment 3 - Ostvarena propusnost sa a) *IntServ* aplikacijom i b) QoS SDN aplikacijom

5.3 SOFTVER KORIŠĆEN U IMPLEMENTACIJI TESTBED-A

Dostupne mogućnosti OpenFlow protokola zavise od njegove verzije. U vrijeme pisanja ovog rada aktuelna verzija protkola je 1.5 [87]. Međutim, izbor OpenFlow kontrolera (koji je trebalo nadograditi) i softverskog *switch-a* usko je povezan sa verzijom protokola. Stoga, ove odluke nije moguće donositi nezavisno.

Trenutno samo Open vSwitch model softverskog *switch*-a omogućava obradu paketa brzinama dovoljno velikim da bi se oponašale realne mreže. Ovo je rezultat činjenjice da je to jedini softverski *switch* sa podrškom za OpenFlow protokol koji radi sa kernel modulom da bi omogućio velike brzine obrade paketa. Implementacije u korisničkom prostoru ne mogu da prate ni približno brzine tog reda. U eksperimentima se pokazalo da su njihova ograničenja na računarima prosječnih performansi reda 20Mb/s. Dakle, samo Open vSwitch se pokazao pogodnim za realizaciju opisanih eksperimenata, što je nametnulo ograničenje korišćenja OpenFlow verizije 1.0. Novije verzije protokola nisu u potpunosti podržane. Sa druge strane Open vSwitch nudi set dodatnih mogućnosti koje su izvan OpenFlow v1.0 specifikacije (*Nicira extensions* [81]) kojima se nedostaci u odnosu na novije verzije u velikom dijelu mogu kompenzovati. Jedna od takvih mogućnosti je iskorišćena za realizaciju modula za rezervaciju ruta, a to je *match* na identifikator pravila u tabeli tokova, čime je omogućeno brisanje rezervacija na ruti nakon isticanja njenog roka važenja.

Kao rešenje za implementaciju kontrolera izabran je POX. U istraživačkim krugovima je do sada najviše korišćen NOX (u C++) kontroler. Međutim, NOX više nije u aktivnoj fazi razvoja i to je glavni razlog zašto je donijeta drugačija odluka. Sa druge strane POX je baziran na Python okruženju koje omogućava brži razvoj aplikacija. Naravno, s obzirom da je Python generalno sporiji od ostalih programskih jezika, ova prednost ostavarena na račun nešto lošijih performansi. Kako je cilj izvršenih eksperimenata bio samo potvrda koncepta, to se nije smatralo velikom preprekom.

Tokom rada na realizaciji *testbed*-a identifikovano je nekoliko nedostataka originalnog dizajna POX kontrolera koje je korisno spomenuti. Jedan od njih je da se čitava kontrolna logika izvršava u jednoj niti. Iako to olakšava implementaciju algoritama, u isto vrijeme ograničava njegovu skalabilnost tako da je veći nivo paralelisma poželjan. Sama Python priroda ovog softvera u mnogome ometa prevazilaženje ovog problema. Sa tog aspekta možda je povoljnija implementacija istog rešenja na više-nitnim kontrolerima kao što je Floodlight. Drugo ograničenje je nedostatak prioritizacije dolaznih poruka. Neke poruke moraju da čekaju prilično vremena da budu obrađene. Ponekad je ovo vrijeme previše veliko, tako da se može desiti da je bafer već pun u trenutku dolaska novih poruka. U eksperimentima se ovo pokazalo naročito kritičnim u pogledu prihvatanja LLDP paketa, čije kašnjenje navodi kontroler na zaključak da je link u kvaru. Na kašnjenje LLDP poruka u originalnom POX dizajnu utiče veliko saobraćajno opterećenje u mreži, usled kojeg može doći i do njihovog gubljenja. Međutim, taj problem može se riješiti kreiranjem specijalnih bafera sa garantovanom propusnošću na OpenFlow *switch*-evima. Nažalost, nedostatak prioritizacije na kontroleru je tipičan i za ostala *open-source* rešenja.

6. ZAKLJUČAK

U današnjim mrežama dominantno su zastupljeni algoritmi rutiranja najkraćom putanjom. U uslovima većeg saobraćajnog opterećenja korisnici se suočavaju sa slabim kvalitetom servisa jer se njihovi saobraćajni tokovi često prepliću na putu do destinacije, dok su sa druge strane mrežni resursi uglavnom neiskorišćeni. Ovo je posebno problematično kada je riječ o multimedijalnim aplikacijama koje zahtijevaju određeni nivo kvaliteta servisa da bi ispravno funkcionisale. Zbog svog značaja ovaj problem je intenzivno analiziran u literaturi. Međutim, predložena rešenja nisu implementirana u praksi zbog ograničenja zastupljene mrežne arhitekture koju karakteriše distribuirana kontrolna ravan. Iz tog razloga, u ovom radu analizirana je mogućnost garancije kvaliteta servisa u SDN mrežama, kod kojih su kontrolna ravan i ravan podataka razdvojene, i kontrolna funkcionalnost prepuštena logički centralizovanom kontroleru. Potencijal ovog koncepta ogleda se u tome što se tehnike za inženjering saobraćaja mogu znatno efikasnije i inteligentnije implementirati u centralizovanom sistemu koji ima globalni pregled stanja u mreži i globalni uvid u zahtjeve aplikacija. Takođe, SDN podrazumijeva programabilnost mrežnih uređaja, tako da se njihovo ponašanje može dinamički mijenjati u cilju optimalne alokacije mrežnih resursa i izbjegavanja zagušenja.

U radu su analizirani centralizovani QoS algoritmi rutiranja sa aspekta njihove pogodnosti za primjenu u SDN/OpenFlow mrežama. Specijalno je razmatran problem garancije propusnosti i kašnjenja u *backbone* mrežama velikih razmjera, gdje je matrica saobraćaja teško predvidljiva. Pored pružanja zahtijevanih performansi servis provajderima, koji iznajmljuju resurse *backbone* mreže, cilj analiziranih rešenja je maksimizacija iskorišćenosti resursa, kao glavni interes provajdera infrastrukture. S tim u vezi, vjerovatnoća blokiranja QoS zahtjeva definisana je kao glavna metrika za procjenu performansi algoritama.

Tehnike za inženjering saobraćaja u *backbone* mrežama koje su do sada predložene u literaturi dominantno su fokusirane samo na garanciju propusnosti. U radu su pomoću simulacija upoređena sva relevantnija rešenja ovog tipa koja ispunjavaju uslove za primjenu u SDN mrežama. Dobijeni rezultati pokazuju da algoritmi koji se ne adaptiraju raspodjeli saobraćaja u mreži uvijek dovode do suboptimalnog iskorišćenja mreže, koje u nekim slučajevima može biti prilično značajno. Konkretno, algoritmi rutiranja najkraćom putanjom i njegove varijacije neravnomjerno troše resurse i ranije od ostalih počinju da blokiraju QoS zahtjeve, dok performanse algoritama koji akcenat stavljaju na balasiranje saobraćaja sa porastom opterećenja naglo opadaju. Pokazalo se da algoritmi koji koriste informacije o položaju ulaznih i izlaznih čvorova najefikasnije koriste mrežne resurse, ali to postižu izborom veoma dugačkih ruta čak i u uslovima slabog saobraćajnog opterećenja. Stoga, iako su ovi algoritmi poželjniji sa aspekta NSP provajdera, nisu prihvatljivi za provajdere servisa osjetljivih na kašnjenje, jer im se ni pri malom opterećenju mreže ne mogu garantovati prihvatljive performanse.

Dodavanjem podrške za garanciju kašnjenja analizirani algoritmi rutiranja se komplikuju toliko da njihovo izvršavanje nije moguće u realnom vremenu. Razlog je što tehnike za inženjering saobraća optimizuju iskorišćenost mrežnih resursa pažljivim odabirom težinskih faktora linkova, koji predstavljaju aditivno ograničenje rutiranja. U kombinaciji sa kašnjenjem koje je takođe aditivno ograničenje, problem izbora ruta postaje *NP-complete*. U radu su opisane aproksimativne tehnike za uprošćavanje ovog problema, međutim, pokazalo se da pri većim brzinama izvršavanja greške usled aproksimacije postaju nedopustivo velike. S obzirom da je jedan od ključnih izazova primjene SDN-a skalabilnost kontrolera, brzina izvršavanja algoritma istaknuta je kao veoma bitan parametar performansi. Iz tog razloga, razmatran je model kontrolnog okruženja u kojem se QoS zahtjevi klasifikuju u konačan broj kategorija. U osnovnom obliku to su kategorija saobraćaja koja nije osjetljiva na kašnjenje (NDS) i kategorija za koju se garantuje kašnjenje ispod određene vrijednosti (DS). Za takav scenario predložen je novi algoritam rutiranja koji podrazumijeva da SDN kontroler pri inicijalizaciji mreže računa set putanja sa kašnjenjem koje je prihvatljivo za DS saobraćaj, a da pri dolasku DS zahtjeva relativno brzo na osnovu određenog kriterijuma bira jednu od njih. Po potrebi DS zahtjevi se mogu diferencirati u više potklasa, tako da je u praksi sa stanovišta korisnika ovaj model prihvatljiv.

Predloženi algoritam baziran je na konceptu minimizacije "interferencije" između parova ulazno-izlaznih čvorova u mreži. Ovaj koncept preuzet je iz literature, ali je u radu oblikovan u kontekstu primjene u SDN okruženju, koje pruža mogućnost podešavanja logike rutiranja samoj aplikaciji. Nedostaci postojećih rešenja baziranih na istom konceptu identifikovani su jednostavnim ilustrativnim primjerima, i shodno tome predložene su potrebne izmjene. Simulacijom je pokazano da u razmatranom scenariju predloženi algoritam, iako relativno jednostavan i računski efikasan, bolje balansira opterećenje linkova i odbacuje manju količinu QoS saobraćaja od konkurentskih, kompleksnih algoritama rutiranja za garanciju kašnjenja i propusnosti.

S obzirom da SDN još uvijek predstavlja tehnologiju u razvoju, dostupna rešenja kontrolera nemaju implementiranu QoS podršku. Kao potencijalno rešenje predložen je i implementiran dizajn OpenFlow kontrolera koji automatski i fleksibilno programira mrežne uređaje tako da se aplikacijama obezbjeđuje zahtijevani nivo kvaliteta servisa. Centralizovana priroda kontrolne ravni SDN arhitekture je iskorišćena za dobijanje informacija o nivou zagušenja u mreži, u skladu sa kojima se biraju rute za *best-effort* saobraćaj. Na ovaj način umanjen je negativni uticaj rezervacije resursa na performanse neprioritetnog saobraćaja. Predstavljeni eksperimentalni rezultati potvrđuju da je predložena QoS arhitektura sposobna da se dinamički prilagođava promjenama u mreži.

Prostor za dalje istraživanje prepoznat je u tehnikama za inženjering saobraćaja u mrežama gdje je SDN samo djelimično zastupljen. Ovaj scenario ima veliki praktični značaj jer omogućava brže prihvatanje SDN-a kao nove tehnologije umrežavanja. Sa druge strane, prednosti centralizovanog pregleda mreže mogu se efikasnije iskoristiti kroz adaptivnu alokaciju propusnog opsega. Osnovna ideja je dinamičko zauzimanje i oslobađanje resursa transportne mreže za različite saobraćajne tokove ili klase, u cilju boljeg odgovora na promjene saobraćaja i pružanja kvalitetnijeg servisa. Na primjer, ukoliko neka klasa generiše

mali nivo saobraćaja u određenom periodu, sve dok su njeni QoS zahtjevi zadovoljeni dio kapaciteta može biti dealociran i stavljen na raspolaganje ostalim servisima ili pripisan dijeljenim resursima. Na ovaj način dodatno se može poboljšati iskorišćenost mreže.

LITERATURA

[1] K. Claffy, G. Miller, K. Thompson, "The nature of the beast: recent traffic measurements from an internet backbone," *Inet98, 1998,* http://www.caida.org/outreach/papers/1998/Inet98/Inet98.html.

[2] *Integrated services in the Internet architecture: An overview*, IETF, RFC 1633, June 1994.

[3] *An architecture for differentiated services,* IETF, RFC 2475, Dec. 1998.

[4] *Multiprotocol label switching architecture,* IETF, RFC 3031, Jan. 2001.

[5] H. E. Egilmez, T. Dane, K. T. Bagci, A. M. Tekalp, "OpenQos: An OpenFlow controller design for multimedia delivery with end-to-end quality of service over Software-Defined-Networks," *APSIPA Annual Summit and Conf.,* Los Angeles, CA, Dec. 2012.

[6] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S. Lee, P. Yalagandula, "Automated and scalable QoS control for network convergence," *INM/WREN''10,* San Jose, CA, Apr. 2010.

[7] Open Networking Foundation, "Software Defined Networking: the new norm for networks," Web. White Paper, Retrieved Apr. 2014.

[8] Urs Hoelzle, *"*OpenFlow at Google*", Open Networking Summit 2012,* 17 April 2012.

[9] M. Kodialam, and T. V. Lakshman. "Minimum interference routing with applications to MPLS traffic engineering," *IEEE INFOCOM 2000,* Vol. 2., 2000.

[10] R. Boutaba, W. Szeto, and Y. Iraqi. "DORA: Efficient routing for MPLS traffic engineering," *Journal of Network and Systems Management10.3,* 2002, pp. 309-325.

[11] S. Suri, M. Waldvogel, and P. R. Warkhede. "Profile-based routing: A new framework for MPLS traffic engineering," *Quality of future Internet Services*, Springer, 2001.

[12] Q. Ma, and P. Steenkiste. "On path selection for traffic with bandwidth guarantees," *IEEE international conference on network protocols*, 1997.

[13] K. Malaoui, A. Belghith, J. Bonnin, and M. Tezeghdanti, "Performance evaluation of QoS routing algorithms," *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, 2005, pp.66, 2005.

[14] C. Fraleigh, F. Tobagi, and C. Diot. "Provisioning IP backbone networks to support latency sensitive traffic," *INFOCOM 2003*, Vol. 1, 2003.

[15] *Simple Network Management Protocol* (SNMP), IETF, RFC 1157, May 1990.

[16] *Automatically Swtiched Optical Netowork (ASON) Routing for OSPFv2 Protocols*, IETF, Jan. 2013.

[17] *Generalized Multi-Protocol Label Switching (GMPLS) Architecture*, IETF, RFC 3945, 2004.

[18] *Transaction Language 1 (TL1) protocol*, Telecorida. Source: *http://telecom-info.telcordia.com/site-cgi/ido/docs.cgi?ID=SEARCH&DOCUMENT=GR-831&, Retrieved: 30 March 2015.*

[19] S. Das, Y. Yiakoumis, G. Parulkar, N. McKeown, P. Singh, P., D. Getachew, P.D. Desai, "Application-aware aggregation and traffic engineering in a converged packet-

circuit network," *National Fiber Optic Engineers Conference on Optical Fiber Communication Conference and Exposition (OFC/NFOEC)*, pp.1-3, 6-10 March 2011.

[20] Y. Chen, T. Farley, and Y. Nong "QoS requirements of network applications on the Internet," *Information, Knowledge, Systems Management* 4.1, pp. 55-76., 2004.

[21] *Specification of the Controlled-Load Network Element Service*, IETF, RFC 2211, 1997.

[22] *Specification of Guaranteed Quality of Service*, IETF, RFC 2212, Sept. 1997.

[23] L, Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, "RSVP: a new resource ReSerVation Protocol," *IEEE Network*, Vol.7, No.5, pp.8-18, Sept. 1993.

[24] *OSPF Version 2*, IETF, RFC 2328, April. 1998.

[25] *OSI IS-IS Intra-domain Routing Protocol*, IETF, RFC 1142, Feb. 1990.

[26] S. Aidarous, and T. Plevyak, "Managing IP networks: challenges and opportunities" Vol. 7, John Wiley & Sons, 2003.

[27] *RSVP-TE: Extensions for LSP Tunnels*, IETF, RFC 3209, Dec. 2001.

[28] S. Das, "A unified control architecture for packet and circuit network convergence," PhD Dissertation, Stanford Univeristy, Department of electrical engineering, June 2012.

[29] *A Border Gateway Protocol 3 (BGP-4)*, IETF, RFC 4271, Jan. 2006.

[30] *BGP/MPLS IP Virtual Private Networks (VPNs)*, IETF, RFC 4364, Feb. 2006.

[31] *LDP Specification*, IETF, RFC 5036, Oct. 2007.

[32] *BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)*, IETF, RFC 4456, April 2006.

[33] *Multiprotocol Extensions for BGP-4*, IETF, RFC 4760, Jan. 2007.

[34] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S, Azodolmolky, S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, Vol.103, No.1, pp.14-76, Jan. 2015.

[35] T. Benson, A. Akella, and D. Maltz, "Unraveling the complexity of network management," *NSDI'09*, Berkeley, CA, USA, 2009, pp. 335–348.

[36] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Intelligent design enables architectural evolution," *10th ACM Workshop on Hot Topics in Networks (ACM HotNets-X)*, New York, NY, USA, 2011, pp. 3-6.

[37] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: Decoupling architecture from infrastructure," *11th ACM Workshop on Hot Topics in Networks*, New York, USA, 2012, pp. 43–48.

[38] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, Vol. 51, No. 2, pp. 114–119, 2013.

[39] J. Pan, S. Paul, and R. Jain, "A survey of the research on future internet architectures," *IEEE Communications Magazine*, Vol. 49, No. 7, pp. 26–36, July 2011.

[40] R. Barrett, S. Haar, and R. Whitestone, "Routing snafu causes internet outage," Interactive Week, 1997.

[41] K. Butler, T. Farley, P. McDaniel, and J. Rexford, "A survey of BGP security issues and solutions," *Proceedings of the IEEE*, Vol. 98, No. 1, pp. 100–122, Jan 2010.

[42] J. Sherry and S. Ratnasamy, “A survey of enterprise middlebox deployments,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-24, Feb 2012.

[43] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, “Onix: a distributed control platform for large-scale production networks,” *9th USENIX conference on Operating systems design and implementation*, Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.

[44] S. Tomovic, M. Pejanovic-Djurisic, I. Radusinovic, "SDN-based mobile networks: concepts and benefits,"*Wireless Personal Communications*, 78(3), 2014, pp. 1629-1644.

[45] S. Tomović, M. Pejanovic-Djurisic, K. Yoshigoe, I. Maljevic, I. Radusinović, "SDN-based concept of QoS aware heterogeneous wireless network operation," *TELFOR*, Nov. 2014.

[46] OpenFlow switch specification v1.0.0, Retrieved Apr. 2014, Available at: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf.

[47] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern, “Forwarding and Control Element Separation (ForCES) Protocol Specification,” IETF, Available: http://www.ietf.org/rfc/rfc5810.txt

[48] H. Song, “Protocol-oblivious Forwarding: Unleash the power of SDN through a future-proof forwarding plane,” *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, New York, USA, 2013, pp. 127–132.

[49] M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher, “OpFlex Control Protocol,” Internet Draft, IETF, April 2014. [Online]. Available: http: //tools.ietf.org/html/draft-smith-opflex-00

[50] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, “OpenState: Programming platform-independent stateful OpenFlow applications inside the switch,” *ACM SIGCOMM Comput. Commun. Rev.*, Vol. 44, No. 2, pp. 44–51, Apr. 2014.

[51] M. Sune, V. Alvarez, T. Jungel, U. Toseef, and K. Pentikousis, “An OpenFlow implementation for network processors,” *Third European Workshop on Software Defined Networks*, 2014.

[52] D. Parniewicz, R. Doriguzzi Corin, L. Ogrodowczyk, M. Rashidi Fard, J. Matias, M. Gerola, V. Fuentes, U. Toseef, A. Zaalouk, B. Belter, E. Jacob, and K. Pentikousis, “Design and implementation of an OpenFlow hardware abstraction layer,” *ACM SIGCOMM Workshop on Distributed Cloud Computing*, New York, USA, 2014, pp. 71–76.

[53] B. Belter, A. Binczewski, K. Dombek, A. Juszczyk, L. Ogrodowczyk, D. Parniewicz, M. Stroinski, and I. Olszewski, “Programmable abstraction of datapath,” *Third European Workshop on Software Defined Networks*, pp. 7-12, 2014.

[54] R. Khondoker, et al. "Feature-based comparison and selection of Software Defined Networking (SDN) controllers," *IEEE World Congress on Computer Applications and Information Systems (WCCAIS)*, pp. 1-7, 2014.

[55] NOX. [Online]. Available at: http://www.noxrepo.org/nox/aboutnox/

[56] POX. [Online]. Available at: http://noxrepo.org/pox/about-pox/

[57] Floodlight. [Online]. Available at: ://www.projectfloodlight.org/
[58] Trema. [Online]. Available at: https://github.com/trema/trema/
[59] OpenDaylight. [Online]. Available at: http://www.opendaylight.org/
[60] Kartik Gopalan, "Efficient provisioning algorithms for network resource virtualization with QoS guarantees," PhD thesis, Stony Brook University, Aug. 2003.
[61] J. Bennett, and Hui Zhang. "WF2Q: worst-case fair weighted fair queueing," *IEEE INFOCOM'96*, Networking the Next Generation, Fifteenth Annual Joint Conference of the IEEE Computer Societies. Vol. 1., pp. 120-128, 1996.
[62] J. Leeuwen (1998): "Handbook of Theoretical Computer Science", Vol. A, Algorithms and complexity, Amsterdam, Elsevier, ISBN 0262720140.
[63] Crawley, R. Nair, B. Rajagopalan, H. Sandick, "A Framwork for QoS-based Routing in the Internet, RFC 2386, 1998.
[64] Z. Wang and J. Crowcroft, "QoS Routing for Supporting Multimedia Applications", *IEEE Journal on Selected Areas in Communications*, Vol. 14, No. 7, pp. 1228–1234, September 1996.
[65] E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs," Numerische Math. 1, pp. 269-271, 1959.
[66] R. Bellman, "On a routing problem", Quarterly of Applied Mathematics 16, pp. 87–90, 1958.
[67] J. M. Jaffe, *"*Algorithms for Finding Paths with Multiple Constraints,*" Networks - International journal*, vol. 14, pp. 95–116, 1984.
[68] Ilmari Juca, "Analysis of Quality of Service Routing Approaches and Algorithms," MSc thesis, Helsinki University of Technology, March 2003.
[69] R.Guerin, A. Orda, D. Williams, "QoS routing mechanisms and OSPF extensions," *IEEE GLOBECOM '97*, vol. 3, pp. 1903-1908, 1997.
[70] G. Apostolopoulos, D. Williams, S. Kamat, R. Guerin, A. Orda, T. Przygienda, "QoS routing mechanisms and OSPF extensions", RFC 2676, 1999.
[71] Wang, Zheng, and Jon Crowcroft. "Routing algorithms for supporting resource reservation." *IEEE JSAC*, 1996.
[72] Kumar, Anurag, D. Manjunath, and Joy Kuri, "Communication networking: an analytical approach," Elsevier, 2004.
[73] A. V. Goldberg, R. E. Tarjan, "Solving Minimum Cost Flow Problem by Successive Approximation", *ACM Symposium on the Theory of Computing*, pp.7-18, 1987.
[74] Z. Wang, J. Crowcroft, "Quality of service routing for supporitng multimedia applications," *IEEE Journal on selected areas in Comm.*,vol. 14. no. 7, pp. 1228-1234, Sep. 1996.
[75] Y. Yang, J.K. Muppala, S.T. Chanson, "Quality of service routing algorithms for bandwidth-delay constrained applications," *Ninth International Conference on Network Protocols*, pp.62-70, 11-14 Nov. 2001.
[76] S. Chen, "Routing Support for providing guaranteed end-to-end Quality-of-Service," PhD thesis, Computer Science, University of Illinois, Urbana-Champaign, 1999.
[77] J. Yen, "Finding the k shortest loopless paths in a network," Management Science 17.1, pp. 712-716., 1971.

[78] S. Tomović, I. Radusinović, "OpenMont: SDN kontroler za garanciju kvaliteta servisa", *ETRAN* 2014, Vrnjačka Banja, Jun 2014.
[79] S. Tomović, N. Prasad, I. Radusinović, "SDN control framework for QoS provisioning", *TELFOR*, Nov. 2014.
[80] *Definitions of Managed Objects for Bridges with Rapid Spanning Tree Protocol*, IETF, RFC 4318, Dec. 2005.
[81] OpenvSwitch. [Online]. Available at: htttp://openvswitch.org
[82] Discussion on Open vSwitch mailinglist. Available at : http://openvswitch.org/pipermail/ discuss/2013-March/009241.html.
[83] Tim Henrinckx, "Dynamic and performance driven control for OpenFlow networks", Master thesis, Dept. of Information Technology, Faculty of Engineering and Architecture, Ghent Univ., June 2013.
[84] Open Networking Foundation, "OpenFlow Management and Configuration Protocol 1.2", Retrieved Sept. 2014, Available at: https://www.opennetworking.org/images/stories/downloads/sdn
[85] J. L. Valenzuela, A. Monleon, I. San Esteban, M. Portoles, O. Sallent, "A Hierarchical tocken bucket algorithm to enhance QoS in 802.11: Proposal, Implementation and Evaluation," *VTC2004-Fall*, vol. 4, pp. 2659-2662, 26-29 Sept. 2004.
[86] Iperf - The TCP/UDP bandwidth measurement tool. [Online]. Available at: http://iperf.sourceforge.net
[87] OpenFlow switch specification v1.5, Retrieved Mach 2015, Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

LISTA SKRAĆENICA

AF - *Assured Forwarding*

ASON - *Automatically Switched Optical Network*

BGP - *Border Gateway Protocol*

CS - *Class Selector*

DDOS - *Distributed Denial-Of-Service*

DiffServ - *Differentiated Services*

DORA - *Dynamic Online Routing Algorithm*

DS - *Delay Sensitive*

E-BGP - *External BGP*

EF - *Expediated Forwarding*

FTP - *File Transfer Protocol*

GMPLS - *Generalized MPLS*

IETF - *Internet Engineering Task Force*

IntServ - *Integrated Services*

IP - *Internet Protocol*

IS-IS - *Intermediate System to Intermediate System*

ISP - *Internet Service Provider*

LDP - *Label Distribution Protocol*

LSR - *Label Switch Router*

MHA - *Minimum Hop Algorithm*

MIRA - *Minimum Interference Routing Algorithm*

MPLS - *Multi-Protocol Label Switching*

NAT - *Network Address Translation*

NDS - *Non-Delay Sensitive*

NP - *Nondeterministic Polynomial time*

NSP - *Network Service Provider*

OSI - *Open Systems Interconnection basic reference model*

OSPF - *Open Shortest Path First*

QoS - *Quality of Service*

RR - *Route Reflector*

RSVP - *Resource Reservation Protocol*

SDN - *Software Defined Networking*

SNMP - *Simple Network Management Protocol*

SPF - *Shortest Path First*

STP - *Spanning Tree Protocol*

SWP - *Shortest Widest Path*

TE - *Traffic Engineering*

TL-1 - *Transaction Language 1*

ToS - *Type of Service*

VLAN - *Virtual Local Area Network*

VPN - *Virtual Private Network*

WAN - *Wide Area Network*

WFQ - *Weighted Fair Queuing*

WSP - *Widest Shortest Path*