UNIVERZITET U NOVOM SADU

FAKULTET TEHNIČKIH NAUKA

RAZVOJ I IMPLEMENTACIJA LBORU METODE ZA DINAMIČKO BALANSIRANJE OPTEREĆENJA SERVERA U HIBRIDNIM SDN MREŽAMA

DOKTORSKA DISERTACIJA

Mentor:

Prof. dr **Živko Bojović**, vanredni profesor

Kandidat:

Teodor Malbašić

Novi Sad, 2022. godine

Zahvalnost

Zahvaljujem se svom mentoru prof. dr Živku Bojoviću na izdvojenom vremenu i svim uloženim naporima u moje doktorske studije. Zahvalio bih se i doc. dr Petru Bojoviću na stručnim i tehničkim savetima tokom istraživanja i izrade doktorske teze. Učešće u pisanju rada za naučni časopis iz kojeg je proistekla ova teza imali su i dr Jelena Šuh, kao i prof. dr Dušan Vujošević.

Posebno bih se zahvalio svojim kolegama i širem radnom kolektivu firme RT-RK, koji su koliko god je bilo moguće izlazili u susret mojim potrebama, i time mi omogućili nesmetano studiranje.

Naročito se zahvaljujem svojoj porodici koja mi je sve vreme pružala, pre svega moralnu, pa i materijalnu podršku i imala razumevanja prema mojim obavezama i ciljevima.

Konačno, zahvalio bih se svim svojim prijateljima na podršci i konstantnom bodrenju u sprovođenju mojih poslovnih ambicija.

УНИВЕРЗИТЕТ У НОВОМ САДУ **ОБРАЗАЦ – 5а**
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА[1]

Врста рада:	Докторска дисертација
Име и презиме аутора:	Теодор Малбашић
Ментор (титула, име, презиме, звање, институција)	Др Живко Бојовић, ванредни професор, Факултет техничких наука
Наслов рада:	Развој и имплементација LBORU методе за динамичко балансирање оптерећења сервера у хибридним SDN мрежама
Језик публикације (писмо):	Српски (латиница)
Физички опис рада:	Унети број: Страница 146 Поглавља 11 Референци 137 Табела 17 Слика 40 Графикона 0 Прилога 1
Научна област:	Електротехничко и рачунарско инжењерство
Ужа научна област (научна дисциплина):	Телекомуникације и обрада сигнала
Кључне речи / предметна одредница:	SDN мрежа, hibridna SDN мрежа, балансирање оптерећења, SNMP протокол, вишепараметарска метрика
Резиме на језику рада:	Софтверски дефинисано умрежавање (SDN) пружа многе предности, укључујући програмирање саобраћаја, агилност, и аутоматизацију рада мреже. Међутим, финансијска ограничења испољена кроз техничке (нпр. скалабилност, толеранција грешака, безбедност) и понекад, пословне изазове (корисниково прихватање и поверење мрежних оператора) чине провајдере несигурним при транзицији на потпуну имплементацију SDN-а. Стога, инкрементална примена SDN функционалности кроз постављање ограниченог броја SDN уређаја међу традиционалне уређаје, представља рационално и ефикасно окружење које клијентима може понудити модерне услуге, са разменом огромне количине података. Међутим, иако хибридна SDN мрежа пружа многе предности, такође поседује и специфичне изазове. Ово истраживање даје одговор на један од ових изазова представљањем истраживања и развоја нове шеме балансирања оптерећења у хибридном окружењу које чини минималан број SDN уређаја (један контролер и један свич). Изложена је нова шема балансирања оптерећења која надзире тренутне индикаторе оптерећења сервера и примењује вишепараметарску метрику при расподели веза како би се балансирало оптерећење сервера на што је могуће ефикаснији начин. Основу нове шеме балансирања оптерећења чини континуално праћење индикатора оптерећења сервера и имплементација вишепараметарске метрике (CPU *load*, I/O *Read*, I/O

[1] Аутор докторске дисертације потписао је и приложио следеће Обрасце:
5б – Изјава о ауторству;
5в – Изјава о истоветности штампане и електронске верзије и о личним подацима;
5г – Изјава о коришћењу.
Ове Изјаве се чувају на факултету у штампаном и електронском облику и не кориче се са тезом.

	Write, Link Upload, Link Download) за распоређивање веза ка серверима. Тестирање обављено на серверима има за циљ што ефикасније балансирање оптерећења сервера. Добијени резултати показали су да се овим механизмом постижу боље перформансе мреже него код постојећих шема балансирања оптерећења у традиционалним и SDN мрежама. Штавише, предложена шема за балансирање може се користити при реализацији разних услуга и применити у било ком клијент-сервер окружењу.
Датум прихватања теме од стране надлежног већа:	01.12.2022.
Датум одбране: (Попуњава одговарајућа служба)	
Чланови комисије: (титула, име, презиме, звање, институција)	Председник: Др Војин Шенк, редовни професор Члан: Др Владо Делић, редовни професор Члан: Др Дејан Вукобратовић, редовни професор Члан: Др Младен Копривица, доцент Члан: Др Петар Бојовић, доцент Члан: Др Живко Бојовић, ванредни професор
Напомена:	

UNIVERSITY OF NOVI SAD
FACULTY OF TECHNICAL SCIENCES

KEY WORD DOCUMENTATION[2]

Document type:	Doctoral dissertation
Author:	Teodor Malbašić
Supervisor (title, first name, last name, position, institution)	Dr Živko Bojović, Associate Professor, Faculty of Technical Sciences
Thesis title:	Development and implementation of the LBORU method for dynamic server load balancing in hybrid SDN networks
Language of text (script):	Serbian language (latin)
Physical description:	Number of: Pages 146 Chapters 11 References 137 Tables 17 Illustrations 40 Graphs 0 Appendices 1
Scientific field:	Electrical and computer engineering
Scientific subfield (scientific discipline):	Telecommunications and signal processing
Subject, Key words:	SDN, Hybrid SDN, Load balancing, SNMP, Multi-parameter metrics
Abstract in English language:	Software-defined networking (SDN) provides many benefits, including traffic programmability, agility, and network automation. However, budget constraints burdened with technical (e.g., scalability, fault tolerance, security issues) and, sometimes, business challenges (user acceptance and confidence of network operators) make providers indecisive for full SDN deployment. Therefore, incremental deployment of SDN functionality through the placement of a limited set of SDN devices among traditional devices represents a rational and efficient environment that can offer customers modern and more data-intensive services. However, while hybrid SDN provides many benefits, it also has specific challenges addressed in the literature. This research answers one of these challenges by presenting the research and development of a new load balancing scheme in the hybrid SDN environment built with a minimal SDN device set (controller and one switch). This dissertation proposes a novel load balancing scheme to monitor current server load indicators and apply multi-parameter metrics for scheduling connections to balance the load on the servers as efficiently as possible. The base of the new load balancing scheme is continuous monitoring of server load indicators and implementations of multi-parameter metrics (CPU *load*, I/O *Read*, I/O *Write*, *Link Upload*, *Link Download*) for scheduling connections. The testing performed on servers aims to balance the server's load as efficiently as possible. The obtained results have shown that this mechanism achieves better results than existing load balancing schemes in traditional and SDN networks.

[2] The author of doctoral dissertation has signed the following Statements:
5б – Statement on the authority,
5в – Statement that the printed and e-version of doctoral dissertation are identical and about personal data,
5г – Statement on copyright licenses.
The paper and e-versions of Statements are held at he faculty and are not included into the printed thesis.

	Moreover, a proposed load balancing scheme can be used with various services and applied in any client-server environment.
Accepted on Scientific Board on:	01.12.2022.
Defended: (Filled by the faculty service)	
Thesis Defend Board: (title, first name, last name, position, institution)	President: Dr Vojin Šenk, Full Professor Member: Dr Vlado Delić, Full Professor Member: Dr Dejan Vukobratović, Full Professor Member: Dr Mladen Koprivica, Assistant Professor Member: Dr Petar Bojović, Assistant Professor Member: Dr Živko Bojović, Associate Professor
Note:	

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

SKRAĆENICE

2G - *2nd Generation*

3G - *3rd Generation*

3GPP - *3rd Generation Partnership Project*

4G - *4th Generation*

5G - *5th Generation*

5GEx - *5G Exchange*

6G - *6th Generation*

API - *Application Programming Interface,* Aplikativna programska sprega

ARP - *Address Resolution Protocol*

BGP - *Border Gateway Protocol*

BGP-LS - *Border Gateway Protocol - Link State*

CIDR - *Classless Inter-Domain Routing*

CPU - *Central Processor Unit,* Centralni procesor

DHCP - *Dynamic Host Configuration Protocol*

DNS - *Domain Name System*

E2E - *End-to-End*

ECU - *Electronic Control Unit,* Elektronska upravljačka jedinica

HTB - *Hierarchy Token Bucket*

HTTP - *HyperText Transfer Protocol*

ICMP - *Internet Control Messaging Protocol*

IEEE - *Institute of Electrical and Electronics Engineers*, Institut inženjera elektrotehnike i elektronike

IKT - Informaciono-komunikacione tehnologije

ILP - *Instruction-level parallelism*, Paralelizam na nivou naredbe

IMEI - *International Mobile Equipment Identity*

IMSI - *International Mobile Subscriber Identity*

IoT - *Internet of Things*

IP - *Internet Protocol,* Internet protokol

IT - *Information Technology*, Informacione tehnologije

I2RS - *Interface to the Routing System*

I/O - *Input-Output,* Ulaz-izlaz

JVM - *Java Virtual Machine*, Java virtuelna mašina

KPI - *Key Performance Indicator*

LAN - *Local Area Network*, Lokalna mreža

LBBSRT - *Load Balancing by Server Response Time*

LBORU - *Load Balancing by Optimizing Resource Utilization*

LC - *Least Connection*

MAC - *Media Access Control*

MIB - *Management Information Base,* Baza upravljačkih informacija

M2M - *Machine to Machine*

NaaI - *Network-as-an-Infrastructure*

NaaS - *Network-as-a-Service*

NFV - *Network Function Virtualization*

OF - *OpenFlow*

OID - *Object Identifier*, Identifikator objekta

OS - *Operating System*, Operativni sistem

OSGi - *Open Service Gateway Initiative*

OSPF - *Open Shortest Path First*

OTT - *Over-The-Top*

OVS - *Open vSwitch*

PCEP - *Path Computation Element Communication Protocol*

PDU - *Protocol Data Unit,* Jedinica podataka protokola

QoS - *Quality of Service,* Kvalitet servisa

QoE - *Quality of experience,* Kvalitet iskustva

RAM - *Random Access Memory,* Radna memorija

RAN - *Radio Access Network*

REST - *Representational State Transfer*

ROS - *Router Operating System*, Operativni sistem rutera

RR - *Round Robin*

RTT - *Round-trip Time*, Vreme povratne putanje

SDN - *Software-defined Networks*, Softverski definisane mreže

SLA - *Service Level Agreement*

SlaaS -*Slice-as-a-Service*

SNMP - *Simple Network Management Protocol*

SOA - Servisno-orijentisana arhitektura

SQL - *Structured Query Language*

TCP - *Transmission Control Protocol*

UDP - *User Datagram Protocol*

UE - *User Equipment*, Korisnička oprema

VLAN - *Virtual LAN*

VM - *Virtual Machine*, Virtuelna mašina

VoD - *Video-on-Demand*, Video na zahtev

VxLAN - *Virtual Extensible LAN*

WLC - *Weighted Least Connection*

WRR - *Weighted Round Robin*

Sažetak

Softverski definisano umrežavanje (SDN) pruža mnoge prednosti, uključujući programiranje saobraćaja, agilnost, i automatizaciju rada mreže. Međutim, finansijska ograničenja ispoljena kroz tehničke (npr. skalabilnost, tolerancija grešaka, bezbednost) i ponekad, poslovne izazove (korisnikovo prihvatanje i poverenje mrežnih operatora) čine provajdere nesigurnim pri tranziciji na potpunu implementaciju SDN-a. Stoga, inkrementalna primena SDN funkcionalnosti kroz postavljanje ograničenog broja SDN uređaja među tradicionalne uređaje, predstavlja racionalno i efikasno okruženje koje klijentima može ponuditi moderne usluge, sa razmenom ogromne količine podataka. Međutim, iako hibridna SDN mreža pruža mnoge prednosti, takođe poseduje i specifične izazove. Ovo istraživanje daje odgovor na jedan od ovih izazova predstavljanjem istraživanja i razvoja nove šeme balansiranja opterećenja u hibridnom okruženju koje čini minimalan broj SDN uređaja (jedan kontroler i jedan svič).

Izložena je nova šema balansiranja opterećenja koja nadzire trenutne indikatore opterećenja servera i primenjuje višeparametarsku metriku pri raspodeli veza kako bi se balansiralo opterećenje servera na što je moguće efikasniji način. Osnovu nove šeme balansiranja opterećenja čini kontinualno praćenje indikatora opterećenja servera i implementacija višeparametarske metrike (CPU *load*, I/O *Read*, I/O *Write*, *Link Upload*, *Link Download*) za raspoređivanje veza ka serverima. Testiranje obavljeno na serverima ima za cilj što efikasnije balansiranje opterećenja servera. Dobijeni rezultati pokazali su da se ovim mehanizmom postižu bolje performanse mreže nego kod postojećih šema balansiranja opterećenja u tradicionalnim i SDN mrežama. Štaviše, predložena šema za balansiranje može se koristiti pri realizaciji raznih usluga i primeniti u bilo kom klijent-server okruženju.

Abstract

Software-defined networking (SDN) provides many benefits, including traffic programmability, agility, and network automation. However, budget constraints burdened with technical (e.g., scalability, fault tolerance, security issues) and, sometimes, business challenges (user acceptance and confidence of network operators) make providers indecisive for full SDN deployment. Therefore, incremental deployment of SDN functionality through the placement of a limited set of SDN devices among traditional devices represents a rational and efficient environment that can offer customers modern and more data-intensive services. However, while hybrid SDN provides many benefits, it also has specific challenges addressed in the literature. This research answers one of these challenges by presenting the research and development of a new load balancing scheme in the hybrid SDN environment built with a minimal SDN device set (controller and one switch).

This dissertation proposes a novel load balancing scheme to monitor current server load indicators and apply multi-parameter metrics for scheduling connections to balance the load on the servers as efficiently as possible. The base of the new load balancing scheme is continuous monitoring of server load indicators and implementations of multi-parameter metrics (CPU *load*, I/O *Read*, I/O *Write*, *Link Upload*, *Link Download*) for scheduling connections. The testing performed on servers aims to balance the server's load as efficiently as possible. The obtained results have shown that this mechanism achieves better results than existing load balancing schemes in traditional and SDN networks. Moreover, a proposed load balancing scheme can be used with various services and applied in any client-server environment.

1. Uvod

Internet je tehnologija prisutna već duže od pola veka. Od samog početka svog postojanja, internet se neprestano širio, stvarajući potrebu za neprekidnim inovacijama u pogledu upravljanja, ali i nadgledanja mrežne infrastrukture, sa ciljem da se održe ili unaprede performanse mrežne infrastrukture kao celine, pa čak i podignu na viši nivo.

U poslednjoj deceniji je prisutan nagli porast implementacije inteligentnih servisa koji značajno utiču na dalji razvoj internet tehnologija. Inteligentni servisi su oni servisi koji se zasnivaju na modernim tehnologijama poput mašinskog učenja, IoT (eng. *Internet of Things*) tehnologija, neuronskih mreža, i ostalih naprednih tehnologija. Ograničenja tradicionalnih protokola komunikacije (TCP/IP protokoli) i postojeće hijerarhijske organizacije mreže sve više utiču na kvalitet servisa (eng. *Quality of Service*, QoS) i na iskustva korisnika (eng. *Quality of experience*, QoE), odnosno njihovo zadovoljstvo pruženom uslugom. Ograničenja se ogledaju u nemogućnosti protokola da ustroje komunikaciju za nove, ali i zahtevnije servise imajući u vidu njihove potrebe sa stanovišta raspoloživih mrežnih resursa. Na primer, provajderi servisa za strimovanje multimedijskog sadržaja, često na samom početku rada nisu bili u mogućnosti da kvalitetno podrže sve zahteve korisnika širom svoje države, a kamoli celog sveta. Ovaj problem se uglavnom rešava kupovinom novih mrežnih resursa, a takav pristup rešavanju problema, ovo istraživanje tumači kao nedostatak adekvatne metode za balansiranje opterećenja mrežnih resursa. Softverski definisane mreže (eng. *Software-defined Networks*, SDN) predstavljaju jedan vid rešenja za ovaj i slične probleme. Naime, SDN mreže omogućavaju konstrukciju (programiranje) korisnički definisanog protokola komunikacije. Definisanjem sopstvenog protokola komunikacije, moguće je prilagoditi mrežu (ili makar jedan njen deo) konkretnom servisu. Ovo je moguće uraditi tako što se protokol komunikacije

definiše na način da to odgovara konkretnom servisu u smislu raspoloživosti njegovih resursa, kao što je npr. server mašina.

Generalno posmatrano, ključna paradigma na kojoj se zasniva SDN tehnologija jeste razdvajanje ravni infrastrukture od ravni upravljanja mrežom [1]. Upravljačka logika se može smestiti na centralni uređaj – SDN kontroler. Primena SDN kontrolera ima za cilj da se omogući prilagođavanje dinamičnosti modernih servisa, jednostavnije upravljanje mrežom, ali i obezbeđivanje boljeg kvaliteta iskustva korisnika [2].

U praksi postoje slučajevi, gde se kupovinom više server mašina proširuju mrežni resursi, kako se performanse sistema ne bi pogoršale, usled povećanja broja istovremenih zahteva korisnika. Ipak, za ovo rešenje su potrebna, pre svega, znatno veća finansijska ulaganja, ali i održavanje većeg broja servera, stoga se ono ne smatra ni približno optimalnim [3]. Umesto toga može se koristiti SDN pristup, koji upravo ima za cilj da omogući efikasnije iskorišćenje raspoložive infrastrukture i obezbedi potreban nivo fleksibilnosti, pa i skalabilnosti u mreži.

Veoma čest problem sa kojim se moderni mrežni servisi suočavaju jeste zagušenje u saobraćanju podataka. Ova doktorska teza bavi se upravo tim problemom. SDN koncept nudi se kao podoban kandidat za rešenje problema. Primenom SDN-a na delu mreže zaduženom za upravljanje saobraćajem podataka koji potiču sa međusobno redundantnih server mašina, moguće je ravnomerno raspodeliti opterećenje među server mašinama [4].

Tradicionalni mehanizmi raspodele opterećenja poput *Random, Round Robin, Weighted Round Robin* ne uzimaju u obzir bilo kakvu meru opterećenja resursa na serverima pri samoj raspodeli. Pored toga, proizvođač mrežne opreme nudi sopstvena (eng. *proprietary*) rešenja. U takvim okolnostima, svaka promena u mreži, zahteva nabavku i integraciju isključivo mrežne opreme istog proizvođača [5]. Da bi se rešili neki od postojećih problema, urađene su određene modifikacije mehanizama za balansiranje opterećenja u mrežama u kojima se primenjuju SDN tehnologije. Primer takve jedne modifikacije jeste balansiranje na osnovu vremena odziva servera (eng. *Load Balancing by Server Response Time*, LBBSRT) [6].

Ovo istraživanje nudi nov mehanizam, LBORU (eng. *Load Balancing by Optimizing Resource Utilization*), čiji je zadatak da periodično prikuplja informacije o opterećenju server mašina, odnosno o zauzetosti fizičkih resursa raspoloživih server mašina. Na osnovu tih informacija, a u skladu sa kompozitnom metrikom koja uzima u obzir prirodu zahteva, vrši se ravnomerna raspodela opterećenja na serverima. Pri tome, LBORU mehanizam je nezavisan od konkretnog vlasnika opreme i infrastrukture, dokle god se koristi otvoreni *OpenFlow* [7] protokol.

Prikupljani su sledeći parametri opterećenja:

- Zauzetost procesora,
- Brzina čitanja sa diska,

- Brzina pisanja na disk,
- Količina preuzetih podataka na *downlink*-u i
- Količina predatih podataka na *uplink*-u.

U toku evaluacije metodologije primenjen je minimalan broj mrežnih komponenti (jedan SDN kontroler, jedan SDN svič [8]). Osim toga, takođe se detaljno porede performanse LBORU metode sa performansama postojećih tradicionalnih mehanizama, ali i LBBSRT mehanizma.

1.1 Predmet istraživanja

Da bi se realizovali ciljevi SDN tehnologije uglavnom se primenjuju sprege (interfejsi) otvorenog standarda i implementiraju različite tehnike virtuelizacije. Uz pad cena računarske opreme, primena pomenutih tehnologija smanjuje troškove realizacije mrežne infrastrukture. Time se stvaraju uslovi za realizaciju veoma heterogenih mrežnih okruženja, u kojima se velikim brzinama razmenjuju ogromne količine različito strukturiranih podataka. Takva mrežna okruženja zahtevaju visok stepen fleksibilnosti mrežne infrastrukture i stvaranje uslova za primenu kompleksnih servisnih arhitektura. *OpenFlow* protokol je upravo razvijen sa namerom da se pojednostavi upravljanje takvom mrežom, smanje troškovi implementacije, obezbedi potreban kvalitet postojećih i svih novih servisa, ali i da se u velikoj meri utiče na kvalitet iskustva korisnika (QoE).

Bez obzira na prednosti koje sa sobom nosi primena SDN tehnologije (primena *OpenFlow* standarda, pojednostavljenje upravljanja i dr.), njena potpuna implementacija odnosno, primena tzv. *full* SDN koncepta nije jednostavna, a kod najvećeg broja mrežnih operatora ni moguća. Zahtevala bi promenu postojeće, tradicionalne infrastrukture, što je moguće samo angažovanjem značajnih finansijskih sredstava, kao i potencijalne prekide u saobraćaju tokom zamene uređaja. Sa stanovišta servis provajdera, ovakav koncept nije prihvatljiv, pa se oni uglavnom odlučuju za inkrementalnu primenu SDN funkcionalnosti. To podrazumeva postepenu zamenu tradicionalne opreme sa SDN uređajima ili tamo gde je to moguće (gde uređaji poseduju tu mogućnost) „dodavanje“ podrške za rad sa *OpenFlow* protokolom. U realnom okruženju, ovakav koncept je poznat pod nazivom hibridna SDN mreža. Realizuje se primenom jednog od sledeća dva rešenja ili njihovom kombinacijom:

- Zamenom dela tradicionalnih mrežnih uređaja sa SDN svičevima i
- Implementacijom hibridnih SDN svičeva koji poseduju SDN, ali i tradicionalne funkcionalnosti *switching*-a, *routing*-a, *firewalling*-a, kao i druge mrežne funkcije.

Hibridne SDN mreže uglavnom se realizuju kroz implementaciju SDN funkcionalnosti u delovima mreže, gde se zahteva veći nivo programabilnosti i dinamička alokacija resursa, kako bi se rešili brojni i sve heterogeniji zahtevi korisnika, koji se najčešće vezuju za inteligentna okruženja.

Dinamička priroda ovakvih okruženja, zahteva, da se između ostalog obezbedi i adekvatna kontrola tokova, efikasnije rutiranje saobraćaja i balansiranje opterećenja (eng. *load balancing*), kako bi se optimalno iskoristili raspoloživi resursi i izbegla eventualna pojava zagušenja. Ovo se, pre svega, odnosi na saobraćaj koji se razmenjuje u *data* centrima, gde je neophodno kreirati fleksibilno mrežno okruženje koje bi se jednostavno prilagođavalo različitim servisnim zahtevima. Važno je napomenuti, da se u *data* centrima danas uglavnom koristi virtuelna računarska infrastruktura, mada se mogu sresti i tradicionalni elementi mreže. Ovim resursima se pridružuju SDN svičevi i time stvara hibridna SDN mreža, u kojoj se mogu implementirati one tehnologije upravljanja saobraćajem, koje nisu primenjive u tradicionalnoj mreži. Uloga SDN svičeva jeste da zavisno od definisane mrežne politike, deo saobraćaja obrađuju prema instrukcijama dobijenim od SDN kontrolera, dok bi drugi deo saobraćaja obrađivali na tradicionalni način.

1.2 Polazne hipoteze

Glavna hipoteza koja će biti ispitana u ovom istraživanju glasi:

Realizacija balansiranja opterećenja na serverima u SDN delu hibridne mreže, primenom višeparametarske metrike treba da doprinese efikasnijem balansiranju opterećenja u odnosu na tradicionalne pristupe poput korišćenja metoda: *Random, Round Robin,* i sl.

Na osnovu definisanog predmeta istraživanja može se izdvojiti nekoliko posebnih hipoteza:

H0.1. Moguće je primeniti SDN funkcionalnosti u delu mreže u cilju podizanja nivoa fleksibilnosti razvoja servisa i dokazivanja pouzdanosti, skalabilnosti i upotrebljivosti SDN koncepta.

H0.2. Moguće je razviti algoritam koji se zasniva na višeparametarskoj metrici, kako bi se optimizovalo, odnosno što ravnomernije raspodelilo opterećenje na serverima u hibridnoj SDN mreži.

Daljim preciziranjem navedenih hipoteza, formulišu se pojedini elementi koji su predmet istraživanja:

H0.1.1. Izgradnjom pouzdane i skalabilne mrežne infrastrukture, moguće je učiniti SDN mrežu takvu da nadilazi po performansama tradicionalne mreže.

H0.1.2. Moguće je smanjiti kompleksnost realizacije infrastrukturnih i drugih servisa primenom hibridne SDN infrastrukture.

H0.2.1. Primena višeparametarske metrike u balansiranju opterećenja servera može da prevaziđe po performansama tradicionalne metode balansiranja opterećenja.

H0.2.2. Uz odgovarajuće valorizacije parametara višeparametarske metrike, podižu se performanse specifičnih mrežnih servisa.

1.3 Ciljevi istraživanja

U definisanju ciljeva istraživanja, pošlo se od činjenice da tradicionalni algoritmi koriste jednostavne metrike, koje ne uzimaju u obzir zauzetost resursa na serverima, te da kao takvi u praksi ne daju željene rezultate. Zato je okosnica ovog istraživanja na primeni SDN tehnologije, kako bi se izgradilo programabilnije mrežno okruženje za potrebe balansiranja opterećenja. Takvo rešenje podrazumeva protokol za prikupljanje podataka sa udaljenog servera koji koriste i tradicionalne mreže, kao što je SNMP (eng. *Simple Network Management Protocol*) mehanizam za prikupljanje informacija o zauzetosti serverskih resursa u realnom vremenu [9]. Dobijene informacije bile bi korišćene kroz implementaciju višeparametarske metrike, što sve zajedno treba da omogući uravnoteženo raspoređivanje pojedinačnih veza. Dakle, ideja je, da se upotrebom SNMP mehanizma maksimalno iskoriste postojeća rešenja za nadgledanje resursa, kako bi se u daljem istraživanju primenom višeparametarske metrike (koju čine: procenat opterećenosti procesora, broj pristupa disku pri čitanju sa diska, broj pristupa disku pri pisanju na disk, količina poslatih podataka preko mreže i količina primljenih podataka mreže) došlo do naprednijeg rešenja za dinamičko balansiranje saobraćaja.

Da bi se izvršila validacija nove šeme dinamičkog balansiranja saobraćaja, neophodno je implementirati adekvatno *testbed* okruženje, isplanirati i sprovesti testiranje. S tim u vezi, biće sagledana primena novog mehanizma balansiranja na primeru servera baza podataka. Razlog za to jeste činjenica, da postojanje širokog spektra različitih upita i struktura baza podataka imaju značajan uticaj na stepen (veličinu) angažovanosti pojedinih resursa fizičke arhitekture, zavisno od vrste upita. Kao primer mogu poslužiti upiti koji:

- Zahtevaju filtriranje ili traženje nestrukturiranih podataka (kao što su string rečenice), čime se značajno opterećuju CPU (eng. *Central Processor Unit*) i RAM (eng. *Random Access Memory*) resursi, ali smanjeno koriste mrežni resursi ili zahtevaju manji broj pristupa disku prilikom čitanja i pisanja na njega.
- Traže podatke iz različitih tabela sa mnogo zapisa, pa se u tom slučaju, resursi za čitanje i pisanje na skladište podataka više koriste od resursa procesora i mreže.
- Nalažu slanje celokupne baze podataka i time prvenstveno opterećuju mrežne resurse.

U okviru istraživanja, postavljeni su sledeći ciljevi:

- Projektovanje i realizacija hibridne mrežne infrastrukture, zasnovane na delimičnoj primeni SDN tehnologije, sa ciljem da se omogući fleksibilna realizacija servisa i efikasna raspodela opterećenja na serverskim resursima i mrežnim linkovima.

- Povećanje efikasnosti, pouzdanosti i skalabilnosti u delu mrežne infrastrukture, korišćenjem softverski definisanih mreža.
- Smanjenje kompleksnosti realizacije infrastrukturnih i drugih servisa primenom hibridne SDN infrastrukture.
- Razvoj modela za evaluaciju mrežne infrastrukture zasnovane na hibridnim SDN mrežama.
- Razvoj programskog rešenja za upravljanje saobraćajem u SDN delu hibridne mreže.
- Razvoj programskog rešenja za prikupljanje i osvežavanje podataka o zauzetosti fizičkih resursa server mašina.

1.4 Metodologija istraživanja

Teorijski deo istraživanja realizovan je prikupljanjem i analizom kako strane tako i domaće literature, u kojoj su sadržani naučni i stručni doprinosi eminentnih stručnjaka iz oblasti koja je predmet istraživanja. Za pregled naučne oblasti koristiće se Cobson usluge, IEEE i EBSCO baze podataka.

Tokom ovog istraživanja kao opšte naučne metode korišćene su metode prikupljanja i analize naučnih rezultata, modelovanje, analitičko-deduktivne i statističke metode. Modelovanje se koristi u razvoju modela SDN dela hibridne mreže na *cloud*-u. Analitičko-deduktivne metode koriste se za analizu podataka dobijenih testiranjem postojećih rešenja za balansiranje opterećenja na serverima, kao i testiranjem rešenja predloženog ovim istraživanjem. Merenje relevantnih parametara i analiza dobijenih rezultata izvršena je pomoću standardnih statističkih metoda.

U eksperimentalnom delu izvršeno je vrednovanje razvijenog modela SDN mrežne infrastrukture. Koncept softverski definisanih mreža koje koriste *OpenFlow* protokol komunikacije integrisan je u infrastrukturu mreže. Sam proces vrednovanja dobijenih rezultata, vršen je sa tehničko-tehnološkog, ali i obrazovnog aspekta. Rezultati dobijeni eksperimentalnim putem, koriste se u cilju potvrde glavne hipoteze o poboljšanju kvaliteta i efikasnosti balansiranja opterećenja u hibridnim SDN mrežama.

Ovde je važno naglasiti, da su rezultati istraživanja predstavljeni tekstualno, opisani i predstavljeni kroz više tabela, slika i dijagrama sa uporednom analizom. Sprovedeno istraživanje ima interdisciplinarno svojstvo, jer uključuje naučne discipline: metodologiju, računarstvo, informatiku, statistiku i druge.

1.5 Struktura i organizacija istraživanja

Uvodni deo ove disertacije definiše predmet istraživanja, polazne hipoteze, ciljeve istraživanja, kao i metodologiju istraživanja. U drugom poglavlju opisuje se analiza postojećih rešenja, opisuje se motivacija za primenu SDN tehnologije umrežavanja, kao i već pomenut SNMP. Treće poglavlje detaljno opisuje najpre softverski definisane mreže, a zatim i teorijske osnove ostalih SDN koncepta primenjenih u ovom istraživanju. Četvrto poglavlje opisuje razvoj modela hibridne SDN mreže za potrebe testiranja modula za balansiranje opterećenja. Potom se u petom poglavlju opisuju tehnologije balansiranja opterećenja u SDN mrežama, uz konkretne primere metoda njegove realizacije. Šesto poglavlje ukratko opisuje nov koncept rešenja za balansiranje opterećenja koji je predmet ovog istraživanja. U sedmom poglavlju detaljno je izložen celokupan proces testiranja, kao i rezultati samog testiranja. Osmo poglavlje daje primer primene hibridnog SDN modela u 6G komunikacijama. U poslednjem poglavlju izneti su zaključci doktorske teze.

U prilogu disertacije, nalaze se detalji implementacije programskog rešenja za modul kojim se vrši balansiranje opterećenja, kao i programskog rešenja za praćenje parametara opterećenja na server mašinama.

2. Analiza postojećih rešenja

Ovo poglavlje ima za cilj, da se u kratkim crtama objasni pojam, uloga i značaj mrežne infrastrukture, sa posebnim osvrtom na izazove/probleme u implementaciji naprednih tehnologija i servisa u tradicionalnim računarskim mrežama sa hijerarhijskom organizacijom. Pronalaženje adekvatnih rešenja, predstavlja glavni motiv koji utiče na razvoj i primenu SDN tehnologije. Dakle, da bi se stekao što precizniji uvid u potrebe za implementacijom SDN funkcionalnosti u okviru mrežne infrastrukture, potrebno je jasno opisati koncept te funkcionalnosti. Imajući u vidu potrebu za inkrementalnom primenom SDN funkcionalnosti, pre svega iz finansijskih razloga, neophodno je ukazati na mogućnost saradnje sa protokolima koji se koriste u mrežama sa tradicionalnom arhitekturom. U tom smislu, na kraju poglavlja opisan je *Simple Network Management Protocol* (SNMP), tradicionalni protokol u IP mrežama, osmišljen sa ciljem nadziranja rada mreže i njenih resursa.

2.1 Pojam i definicija infrastrukture mreže

Mrežnu infrastrukturu čine i fizička arhitektura i programska rešenja koji zajedničkim delovanjem omogućavaju povezanost, komunikaciju, operacije i upravljanje mrežom. Ona obezbeđuje komunikaciju između mrežnog servisa i njegovih korisnika.

Mrežna infrastruktura je najčešće deo jedne šire IT infrastrukture u sastavu IT okruženja mnogih kompanija. Sve komponente infrastrukture mreže međusobno su povezane, te su pored raznih spoljnih komunikacija sa drugim mrežama omogućene i unutrašnje komunikacije.

Entiteti umrežavanja su:

- Ruteri,
- Svičevi,
- Bežični ruteri,
- LAN kartice i
- Kablovi.

Ruteri, svičevi, kao i bežični ruteri mogu biti implementirani i programski, te su onda virtuelni, a ne fizički uređaji.

Tipični programi za umrežavanje su:

- Operativni sistemi,
- Programi za mrežne operacije i upravljanje mrežom,
- Aplikacije za bezbednost
- i drugi.

2.1.1 Tradicionalni hijerarhijski model organizacije mreže

Tradicionalne mreže sa hijerarhijski organizovanom mrežnom arhitekturom, nisu u stanju da adekvatno odgovore na nagli porast broja korisnika, ni na zahteve novih i kompleksnijih servisa, kao ni da efikasno realizuju nove vidove komunikacija (M2M, IoT). Ove mreže čine tradicionalni mrežni uređaji (svičevi i ruteri), sa tradicionalnim L2/L3 IP protokolima komunikacije, a njihov referentni model jeste TCP/IP [10].

Trenutno stanje u mreži dodatno je opterećeno prisustvom sve većeg broja različitih uređaja i proizvođača, koji u najvećoj meri koriste *proprietary* rešenja. To uzrokuje probleme u funkcionisanju mreže, koji se odnose na:

- Kompleksnost upravljanja mrežnom infrastrukturom - zahteva se poznavanje velikog broja mrežnih protokola komunikacije i konfiguracija većeg broja uređaja da bi se implementirao jedan servis, a prisutna su i brojna nestandardna rešenja sa izraženim problemom interoperabilnosti u radu opreme različitih proizvođača (proces standardizacije protokola traje veoma dugo).
- Postojanje velikog broja funkcionalnosti definisanih u fizičkoj arhitekturi - utiče na visoke troškove realizacije infrastrukture i nedovoljan stepen fleksibilnosti, što usporava proces uvođenja novih aplikacija i servisa.

2.1.2 Problemi u tradicionalnim mrežama sa hijerarhijskom organizacijom

Kao jedno od rešenja za nedostatke tradicionalnih mreža, nameće se razvoj i implementacija tehnologija, koje se pre svega oslanjaju na „razdvajanje" hardvera od softvera, tj. primenu rešenja u domenu softverskih tehnologija. Ključni aspekt razvoja ovih tehnologija vezan je za stvaranje uslova, kojim bi se eliminisala zavisnost implementacije servisnih aplikacija od fizičke arhitekture. Cilj je da se razvije skalabilna i adaptibilna mrežna infrastruktura koju bi karakterisali visok stepen dostupnosti, efikasnosti, ali i bezbednosti aplikacija i servisa [11]. Posmatrano sa tog aspekta, tehnologija softverski definisanih mreža, kojom se ravan upravljanja (eng. *control plane*) odvaja od ravni podataka (eng. *data plane*), nametnula se kao prirodno rešenje. Razdvajanjem ove dve ravni, dobija se mogućnost centralizacije celokupne kontrolne logike u jednoj tački - SDN kontroleru. Zato ovakva mrežna arhitektura i predstavlja novu mrežnu paradigmu, jer unosi potreban nivo programabilnosti i pruža efikasnu podršku dinamičkim promenama u mrežnoj infrastrukturi (pre svega u pogledu dinamičke alokacije resursa u skladu sa zahtevima korisnika) [12]. Dodatna prednost jeste to, što se kreira dinamička mrežna arhitektura bez potrebe da sami korisnici poznaju mrežne tehnologije i fizičku topologiju mreže. Centralizacijom kontrolnih funkcija stvaraju se uslovi za obezbeđivanjem dinamičkog kvaliteta servisa u mreži i to kroz:

- Kontinuitet u praćenju performansi mrežne infrastrukture (pojedinačno i u celini),
- Efikasnu detekciju neželjenih situacija u radu mreže i preusmeravanje saobraćaja bez uticaja na servise korisnika,
- Optimalno rutiranje saobraćaja i balansiranje opterećenja na aplikativnom nivou.

2.2 *Simple Network Management Protocol*

Simple Network Management Protocol (SNMP) predstavlja standardizovan internet protokol koji omogućava prikupljanje i izmenu podataka o uređajima povezanim na IP mreže. Prvobitna namena protokola bila je upravljanje i konfiguracija mrežnih uređaja poput modema, rutera, svičeva, servera, itd. Na ovaj način SNMP nalazi svoju primenu u upravljanju i nadzoru mreže.

Podaci kojima SNMP upravlja nalaze se u bazi koja se naziva bazom upravljačkih informacija (eng. *Management Information Base*, MIB). Podaci ove baze podataka opisuju stanje i konfiguraciju sistema smeštenog na jednom uređaju, a SNMP protokol nudi mogućnost njihovog čitanja i izmene.

Tipično je da su učesnici SNMP protokola komunikacije jedan ili više upravljačkih čvorova mreže, čiji je zadatak da nadziru i upravljaju grupom drugih mrežnih uređaja. Na svakom uređaju kojim upravlja SNMP protokol, nalazi se instaliran softver koji se naziva agentom ovog protokola i

njegov zadatak jeste da pruža povratne informacije o prethodno pomenutom upravljačkom uređaju (čvoru).

Tri komponente ulaze u sastav tipične mrežne arhitekture kojom upravlja SNMP protokol:

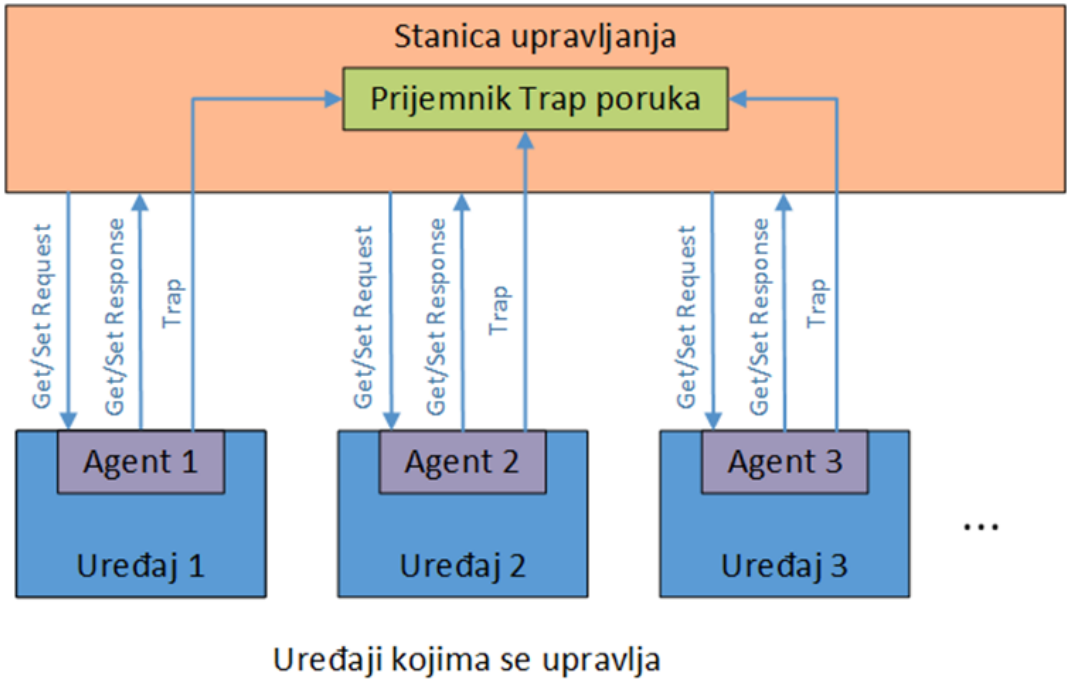
1. Uređaji kojima se upravlja,
2. Agent, softver koji se izvršava na istim uređajima i
3. Stanica za upravljanje mrežom i softver koji se izvršava na upravljačkim uređajima.

Uređaj kojim se upravlja obično nudi takvu spregu sa SNMP protokolom, da je njom omogućena i komunikacija u jednom smeru (samo čitanje podataka) i u oba smera (čitanje i pisanje).

Agent je programsko rešenje koje ima pristup informacijama kojima se može upravljati SNMP protokolom i ima sposobnost da prevodi informacije u SNMP format, kao i iz SNMP formata.

Stanica za upravljanje mrežom ima mogućnost nadzora i upravljanja drugim mrežnim uređajima.

Na slici 2.1 prikazana je ilustracija SNMP komunikacije, na kojoj su predstavljene prethodne tri komponente.



Slika 2.1 SNMP komunikacija

Ovde je bitno istaći da je u toku realizacije tehničkog rešenja ovog istraživanja došlo do odstupanja od prethodno navedenih pravilnosti tipične mrežne arhitekture. Konkretno, komponente 1. i 2. se više ne odnose na iste uređaje. Uređaj kojim se upravlja jeste SDN svič, dok su *database* serveri uređaji na kojima se izvršava agent. Stanicu za upravljanje mrežom predstavlja SDN

kontroler, a softver koji se izvršava na njemu jeste aplikacija implementirana kroz *POX* kontroler (tip SDN kontrolera o kojem će biti više reči kasnije), i njom se vrši balansiranje opterećenja server mašina.

Zašto je došlo do ovih odstupanja? Tipična mrežna arhitektura nalaže da se SNMP protokolom vrši i nadzor i upravljanje mrežom. Tehničko rešenje ovog istraživanja primenjuje SNMP protokol samo u funkciji nadzora mreže, odnosno resursa njenih serverskih komponenti. Upravljanje mrežom, s druge strane, vrši se primenom SDN koncepta i *OpenFlow* protokola komunikacije kojim se upravljaju tokovi podataka i to manipulacijom tabele tokova smeštene na SDN sviču. Drugim rečima, SNMP protokol ne učestvuje (kao što je to slučaj sa tradicionalnim rešenjima) u samom upravljanju saobraćaja u mreži, već ima funkciju isključivo nadzora server mašina, odnosno njihovih fizičkih resursa (procesora, trajne memorije, itd.).

2.2.1 Baza upravljačkih informacija (MIB)

Prisustvo baze upravljačkih informacija je ključno za rukovanje podacima koji se prenose SNMP protokolom. Ove informacije su agentima dostupne u vidu promenljivih. Izmenom ovih promenljivih menjaju se i informacije u samoj bazi. Postoji čitava hijerarhija kada je u pitanju organizacija ovih podataka i zahvaljujući njoj je i ostvaren pristup podacima. Postoje bar dva pozitivna ishoda ove organizacije promenljivih [9]:

1. Ograničava se broj potrebnih funkcija za upravljanje podacima na dve:
    a. upisivanje u promenljivu radi izmene konfiguracije
    b. dobavljanje informacije radi očitavanja stanja ili konfiguracije
2. Izbegava se potreba za uvođenjem imperativnih komandi za upravljanje; primena ovakvih komandi u praksi raste, a semantika kojom se one realizuju je uglavnom proizvoljne složenosti.

Kad je u pitanju prvi pozitivan ishod, nadgledanje mreže na što detaljnijem nivou je ključna karakteristika SNMP protokola. Zato je neophodno da i povlačenje upravljačkih informacija bude ostvareno na što ekonomičniji način, kako se ne bi bespotrebno generisao dodatni saobraćaj koji bi mogao smanjiti performanse celog sistema. SNMP protokol uspešno realizuje povlačenje samo neophodnih informacija pri nadzoru uređaja.

Izbegavanjem imperativnih komandi ostvaruje se smanjena kompleksnost rešenja za SNMP protokol komunikacije. Ne gubi se na funkcionalnosti jer se velika većina naredbi može ostvariti slanjem i upisivanjem odgovarajućeg podatka na uređaj kojim se upravlja SNMP komunikacijom. To se ostvaruje tako što se modifikacijom vrednosti date promenljive pokreće odgovarajuća aktivnost na upravljanom uređaju. Na primer, umesto implementacije komande za restartovanje sistema, moguće

je to učiniti slanjem podatka o vremenu (broju sekundi) nakon kojeg je potrebno odraditi restart sistema.

Baza upravljačkih informacija opisuje strukturu upravljačkih podataka na jednom podsistemu nekog uređaja. Svaka instanca bilo kog tipa podatka definisanog u MIB-u identifikovana je u SNMP operacijama jedinstvenim imenom promenljive. Uopšteno govoreći, ime SNMP promenljive jeste IDENTIFIKATOR OBJEKTA (eng. *Object Identifier*, OID) u obliku x.y, gde je x ime neagregatnog tipa objekta definisanog u MIB-u, a y je fragment IDENTIFIKATORA OBJEKTA koji na način specifičan za imenovani tip objekta, identifikuje željenu instancu. Dakle, OID identifikator služi za identifikaciju promenljive čija se vrednost može iščitati ili upisati preko SNMP protokola.

Primeri OID identifikatora parametara trajne memorije (diska) *linux* mašine:

- Putanja na kojoj je *mount*-ovan disk: .1.3.6.1.4.1.2021.9.1.2.1
- Putanja predviđena podeli diska na particije: .1.3.6.1.4.1.2021.9.1.3.1
- Ukupna veličina diska (kBytes): .1.3.6.1.4.1.2021.9.1.6.1
- Slobodan prostor diska: .1.3.6.1.4.1.2021.9.1.7.1
- Zauzet prostor diska: .1.3.6.1.4.1.2021.9.1.8.1
- Procenat zauzetosti diska: .1.3.6.1.4.1.2021.9.1.9.1
- Procenat *inode*-ova on disku: .1.3.6.1.4.1.2021.9.1.10.1

Prethodno navedeni primeri mogu se pronaći na sajtu [13]. Grafički prikaz podatka o putanji na kojoj je *mount*-ovan disk: .1.3.6.1.4.1.2021.9.1.2.1 prikazan je na slici 2.2.

Slika 2.2 Primer OID identifikatora [13]

Opisi elemenata sa prethodne slike su sledeći:

- 1 iso - *International Organization for Standardization* (ISO)
- 3 identified-organization, org, iso-defined-oganization - Šeme identifikacije organizacije registrovane prema ISO/IEC 6523-2
- 6 dod - *Open System Interconnection* (OSI) mreža *Department of Defense USA*
- 1 internet - internet
- 4 private - privatni projekti
- 1 enterprise, enterprises - privatna preduzeća
- 2021 ucadavis - definiše privatne SNMP MIB ekstenzije
- 9 dskTable - provere diska

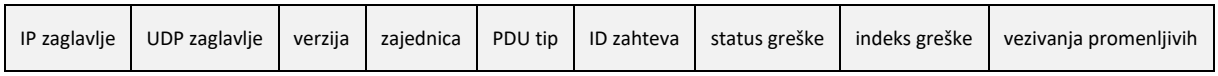
- 1 dskEntry - SNMP provera diska preko interneta
- 2 dskPath - putanja gde je disk *mount*-ovan
- 1 - putanja gde je disk *mount*-ovan

2.2.2 Detalji SNMP protokola

SNMP protokol radi na aplikativnom sloju, a sve poruke se prenose preko UDP protokola. Agent prima poruke na određenom UDP portu (port 161), dok stanica za upravljanje mrežom šalje sve svoje zahteve sa bilo kog svog porta na jedan isti port agenta. Agent potom šalje poruku odgovora nazad istom portu stanice upravljanja. Stanica upravljanja ima definisan port (obično 162) na kojem prihvata poruke obaveštenja od agenta. Agent, sa druge strane, može generisati i slati poruke obaveštenja sa bilo kog slobodnog porta.

PDU paketi

Prva verzija protokola SNMPv1 definiše 5 jedinica podataka protokola (eng. *Protocol Data Units*, PDUs). Dve dodatne jedinice (*GetBulkRequest* i *InformationRequest*) dodate su u verziji SNMPv2. Postoji i treća verzija SNMPv3, ali ona se više neće pominjati, jer je prilikom izrade tehničkog rešenja ovog istraživanja bilo dovoljno primeniti SNMPv2. Svi tipovi SNMP PDU paketa imaju sledeće zaglavlje paketa:



Slika 2.3 Zaglavlje PDU paketa

U nastavku su navedeni svih sedam tipova PDU paketa:

1. *GetRequest* – Zahtev stanice upravljanja, upućen agentu, da se isporuče podaci o vrednostima promenljivih. Poslednje polje paketa, vezivanja promenljivih, specificira o kojim promenljivima je reč. Iste promenljive se dobavljaju atomskim operacijama od strane agenta. Paket tipa *Response*, koji sadrži trenutne vrednosti promenljivih šalje se nazad stanici.
2. *SetRequest* - Zahtev stanice upravljanja, upućen agentu, da se upišu podaci o vrednostima promenljivih. Vezivanja promenljivih zapisana su u sadržaju (telu) paketa zahteva. Izmene svih navedenih promenljivih izvedene su atomskim operacijama agenta. Paket tipa *Response*, koji sadrži nove vrednosti promenljivih šalje se nazad stanici.

3. *GetNextRequest* - Zahtev stanice upravljanja, upućen agentu, za otkrivanjem dostupnih promenljivih i njihovih vrednosti. Vraća se *Response* paket sa vezivanjem promenljive koja predstavlja leksikografski sledeću na redu promenljivu u MIB bazi. Primenom *GetNextRequest*, moguće je celu MIB bazu proslediti stanici. Redovi MIB tabele mogu bit pročitani navođenjem OID identifikatora kolona u polju vezivanja promenljivih paketa.
4. *GetBulkRequest* - Zahtev stanice upravljanja, podseća na višestruke *GetNextRequest*. Ovo je optimizovana verzija *GetNextRequest* PDU paketa, zahvaljujući kojoj se postiže da *Response* paket sadrži više vezivanja promenljivih, polazeći od jednog vezivanja iz zahteva. Postoje posebna polja PDU paketa kojima se upravlja odgovorom. Ova vrsta paketa bila je uključena SNMPv2 verzijom protokola.
5. *Response* - Paket odgovora agenta na zahteve stanice upravljanja (*GetRequest, SetRequest, GetNextRequest, GetBulkRequest, InformRequest*). Sadrži polja statusa greške i indeksa greške, kojima agent odgovara na greške nastale u komunikaciji.
6. *Trap* - Asinhrona poruka agenta, upućena stanici upravljanja. Za razliku od ostalih PDU poruka, gde stanica upravljanja eksplicitno traži od agenta informacije, u ovom slučaju agent šalje PDU poruku bez ikakvog prethodnog zahteva. Ovaj vid komunikacije omogućava agentu da javi stanici upravljanja o informacijama (nekad i nepredviđenim) kao što je na primer vreme trajanja rada sistema.
7. *InformationRequest* - Asinhrona poruka kojom agent obaveštava stanicu upravljanja o prethodno pristiglom PDU paketu.

Veličina paketa PDU poruke ograničena je na manju od naredne dve veličine [14]:

1. Najveća veličina paketa koju primalac poruke može da prihvati
2. Najveća veličina paketa koju pošiljalac poruke može da generiše

Fragmentacijom ili deobom paketa dolazi do povećanja verovatnoće gubitka informacija (paketa) pri njihovom prenosu. Iako su dobrodošle implementacije protokola takvog da je podržano slanje što većih paketa, nikako nije poželjno generisati toliko velike pakete da ih primalac ne može primiti. Zbog toga bi implementacija *GetBulkRequest* paketa trebalo da pripazi na broj poruka koje se razmenjuju protokolom, odnosno da pazi pri minimizaciji tog broja, naročito kada je u pitanju ogromna veličina informacija upravljanja koje se razmenjuju između agenta i stanice upravljanja. Kako bi se to postiglo, paketi se konstruišu tako da budu što veći, ali da takođe istovremeno ispoštuju maksimalnu dozvoljenu veličinu paketa.

Pored toga, potrebno je paziti i na to da veličina PDU paketa po mogućnosti ne prevazilazi veličinu koju podržava najduži mogući put paketa. Dakle, svi usputni čvorovi na putanji od uređaja sa agentom do stanice upravljanja trebalo bi da se drže prethodno navedenih principa.

Obrada PDU paketa zahteva ispravnu sintaksu i kodovanje samo onih polja paketa koja su od interesa proceduri obrade. Ukoliko neka od ostalih polja paketa nisu pomenuta prilikom njegovog kreiranja, SNMP protokol to jednostavno ignoriše i ne nastaje problem pri prijemu paketa. Na primer, postoje tipovi PDU paketa, kao što je *GetRequest*, kojima nije od važnosti vrednost promenljive, već samo njen naziv. Deo paketa u kojem je navedena vrednost promenljive ignoriše se pri obradi.

SNMP *engine*

Entiteti mreže sa podrškom SNMP protokola koriste SNMP *engine* [15] kako bi obavljali konkretne zadatke. Aplikacija SNMP entiteta zadužena je za obradu operacija nad informacijama upravljanja i može koristiti SNMP poruke za komunikaciju sa ostalim SNMP entitetima. SNMP *engine* pruža mogućnosti slanja i primanja poruka, autentifikacije i zaštitnog kodovanja poruka i upravljanja pristupom i objektima upravljanja. SNMP *engine* dovodi se u direktnu vezu sa konkretnim SNMP entitetom koji ga poseduje.

Engine sadrži četiri komponente:

1. Dispečer,
2. Podsistem za obradu poruka,
3. Podsistem za bezbednost i
4. Podsistem za upravljanje pristupom.

Uvek postoji samo jedan dispečer u okviru jednog SNMP *engine*-a. On omogućava istovremenu podršku više verzija SNMP poruka u jednom SNMP *engine*-u i to zahvaljujući:

- Slanju i primanju SNMP poruka ka mreži i od mreže,
- Određivanju verzije SNMP poruke i interakciji sa odgovarajućim modelom za obradu poruka,
- Pružanju apstraktne sprege sa SNMP aplikacijama radi dostave PDU paketa aplikaciji i
- Pružanju apstraktne sprege sa SNMP aplikacijama koja omogućava slanje PDU paketa udaljenom SNMP entitetu.

Podsistem za obradu poruka odgovoran je za pripremu poruka za slanje, kao i za izdvajanje podataka iz primljenih poruka. Svaki model obrade poruka definiše format određene verzije SNMP poruke i kao takav upravlja pripremom i izdvajanjem podataka iz specifične verzije poruke.

Podsistem za bezbednost obezbeđuje usluge bezbednosti poput autentifikacije i privatnosti poruka. Protokol bezbednosti određuje mehanizme, procedure i MIB objekte, primenjene za potrebe autentifikacije i privatnosti.

Podsistem za upravljanje pristupom obezbeđuje usluge autorizacije preko jednog ili više modela upravljanja pristupom. Model upravljanja pristupom definiše konkretnu funkciju odluke o pristupu, kako bi bila implementirana podrška pri donošenju odluke o pravima pristupa.

SNMP kontekst

SNMP kontekst [15], ili skraćeno samo kontekst, predstavlja kolekciju informacija upravljanja kojoj jedan SNMP entitet ima pravo pristupa. Kao što jedan predmet informacije upravljanja može postojati u više konteksta, tako i SNMP entitet može imati pravo pristupa mnogim kontekstima.

Obično postoji mnogo instanci svakog od tipova objekata upravljanja u okviru jednog domena upravljanja. U cilju jednostavnije implementacije metode za identifikaciju instanci definisanih u MIB bazi, nije dozvoljeno da se svaka instanca logički razdvaja od svih ostalih instanci u sastavu jednog istog domena upravljanja. Umesto toga, za svaku instancu je dozvoljena identifikacija samo u okviru nekog konteksta, pri čemu jeste dozvoljeno da jedan domen upravljanja poseduje više konteksta. Kontekst može da predstavlja fizički ili logički uređaj, pa čak i da obuhvata više uređaja, ili podskup na nivou jednog uređaja, kao i podskup na nivou više uređaja. Bitno je da je kontekst uvek definisan kao podskup nad jednim SNMP entitetom.

Dakle, za identifikaciju predmeta informacije upravljanja nisu dovoljni samo tip i instanca objekta upravljanja, već i njegovi identifikatori konteksta i *engine*-a. S druge strane, kombinacija identifikatora konteksta i *engine*-a nedvosmisleno identifikuje kontekst koji pripada jednom administrativnom domenu.

Na primer, tip objekta upravljanja ifDescr definisan je kao opis mrežnog interfejsa. Kako bi se identifikovao opis prvog mrežnog interfejsa uređaja-X, potrebno je pet podataka:

1. Dispečer,
2. Identifikator *engine*-a SNMP entiteta, obezbeđuje pristup informacijama upravljanja na uređaju-X,
3. Naziv (identifikator) konteksta – „uređaj-X“,
4. Tip objekta upravljanja – ifDescr i
5. Instanca objekta upravljanja – „1“.

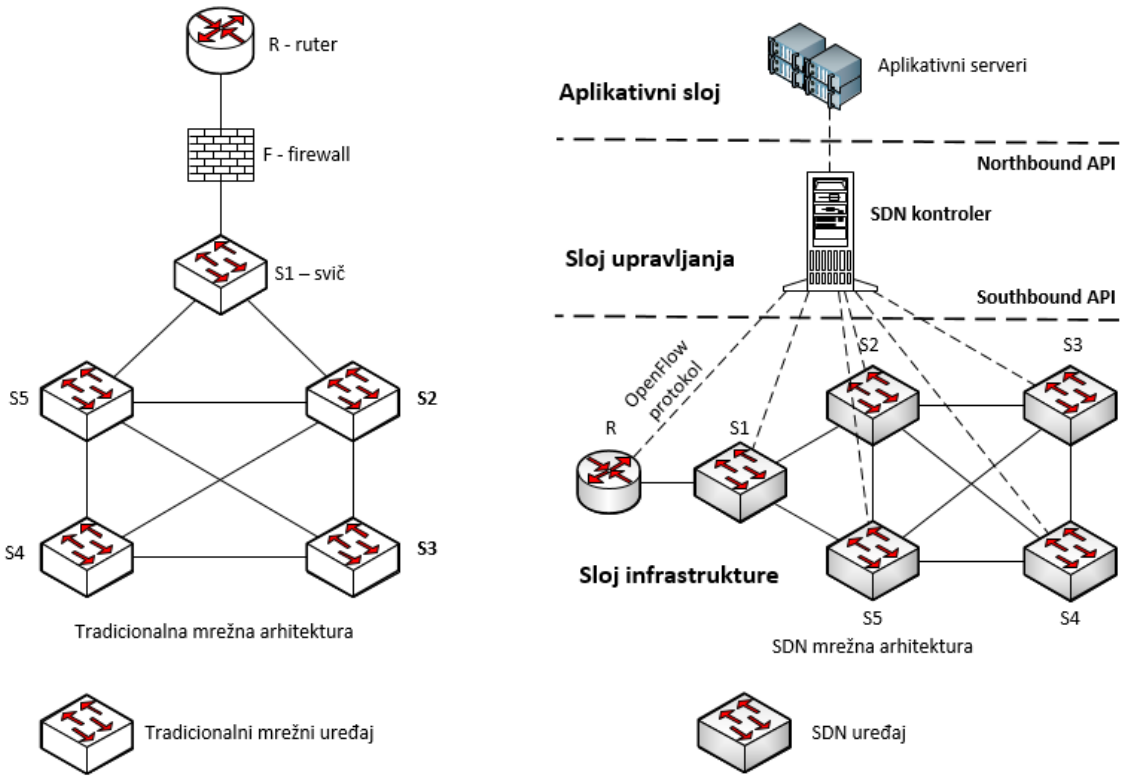
U prilogu disertacije, nalazi se opis programskog rešenja LBORU metode koja primenjuje upravo SNMP protokol u funkciji nadzora opterećenja na server mašinama.

3. Koncept softverski definisanih mreža

Koncept SDN mreže ima za cilj rešavanje više tehničkih problema. Kompleksnost upravljanja mrežom, a u nekim slučajevima i kašnjenje (*network latency*) može se umanjiti uvođenjem SDN koncepta. SDN koncept kao prednost pruža i veći stepen pouzdanosti i programabilnosti, utičući pozitivno i na skalabilnost [1].

Na slici 3.1 prikazana je troslojna struktura SDN mreže [16], a tri sloja su:

1. Aplikativni sloj,
2. Sloj upravljanja i
3. Sloj infrastrukture/podataka.

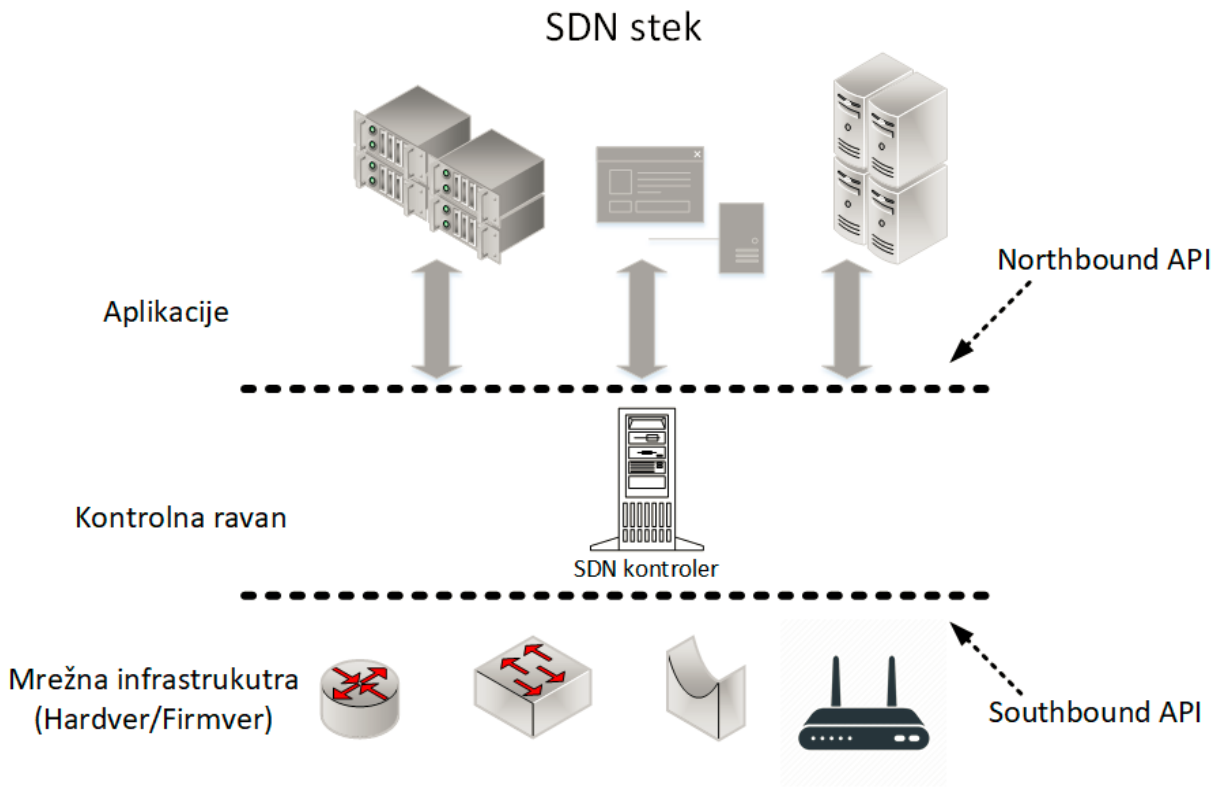


Slika 3.1 Tradicionalna (levo) i SDN (desno) mrežna arhitektura

Generalno posmatrano postoje dva razloga, zbog kojih je na slici 3.1 (desno) SDN funkcionalnost izmeštena na kontroler [17]. Jedan razlog jeste centralizovano upravljanje mrežom. Radi jednostavnije i ekonomičnije implementacije rešenja, upravljanje saobraćajem u SDN mrežama se premešta na SDN kontroler, umesto da se izvršava na svakom pojedinačnom mrežnom uređaju. Ovim se brzina prenosa podataka kroz mrežu znatno uvećava u odnosu na tradicionalne mreže sa distribuiranom kontrolnom ravni (u tradicionalnim mrežama svaki ruter analizira informacije iz paketa i na osnovu destinacione adrese traži optimalnu putanju do odredišta, dok se u SDN mrežama značajan deo analize prepušta kontroleru). Drugi razlog je oslobađanje jednostavnijih mrežnih uređaja (poput rutera i svičeva) od jednog dela obrade podataka koji novi definisani protokoli komunikacije donose sa sobom. SDN kontroler se uglavnom implementira na moćnijim mašinama sa stanovišta procesorske snage, memorijskih i drugih resursa. To jasno ukazuje, da se najveći deo postojećih, kao i primena novih funkcionalnosti kojima se unapređuje efikasnost komunikacije upravo na njemu i implementira.

U aplikativnom sloju se nalaze serveri raznih mrežnih aplikacija. Na ovim aplikativnim serverima se realizuju različiti servisi koji se nude korisnicima. Raspodelu saobraćaja ovih aplikacija, u skladu sa informacijama o raspoloživosti pojedinih mrežnih resursa, a na osnovu servisnih zahteva obavlja SDN kontroler. SDN kontroler predstavlja kontrolni sloj, odnosno centralnu tačku u implementaciji SDN koncepta u mrežama i zadužen je da na centralizovan način upravlja

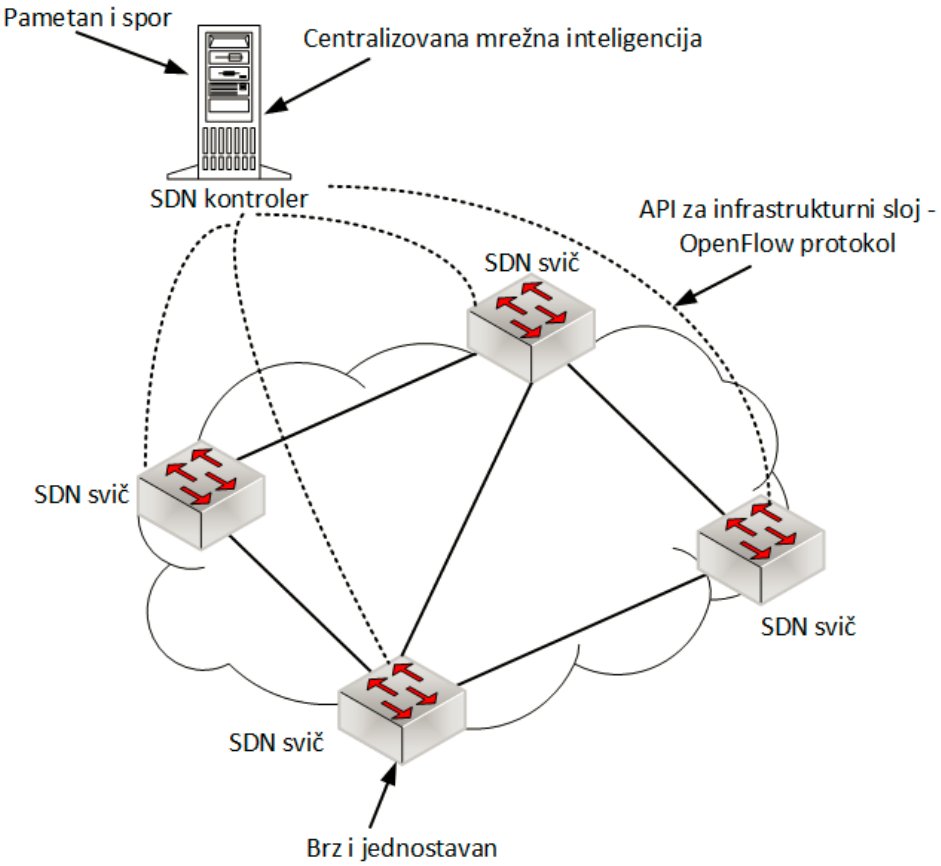
saobraćajem u mreži. Njegov zadatak je da korišćenjem protokola otvorenog standarda (na primer *OpenFlow*), razmenjuje upravljačke poruke sa mrežnim uređajima na sloju infrastrukture. Sprega između server mašina i SDN kontrolera je realizovana kroz takozvani *northbound* API (eng. *Application Programming Interface*), dok se mrežnom infrastrukturom (svičevima i ruterima) može upravljati preko *southbound* API-ja, kao što je i prikazano na slici 3.2.



Slika 3.2 Prikaz SDN steka

Ovde je potrebno naglasiti, da se obe grupe interfejsa (*nothbound* i *southbound* interfejs) uglavnom implementiraju u otvorenom standardu (*OpenFlow*, REST, i sl.). Drugim rečima, sama ideja SDN koncepta realizuje se kroz primenu ovakvih interfejsa koji bi obezbeđivali punu interoperabilnost i nezavisnost od proizvođača opreme, ali i da se i upravljanje mrežom učini lakšim i jednostavnijim.

Dakle, primenom SDN koncepta, u računarskim mrežama se implementira nov način upravljanja koji ima za cilj da se razdvoji proces upravljanja od procesa prosleđivanja i izvrši logička centralizacija mrežne inteligencije u SDN kontroleru (Slika 3.3 Centralizacija mrežne inteligencije).



Slika 3.3 Centralizacija mrežne inteligencije

Sa slike 3.3, lako se može zaključiti, da SDN kontroler ima zadatak da izgradi i održava globalnu sliku o mreži i omogući jednostavnije donošenje odluka. Razlog za to leži u činjenici, da se odluke donose na osnovu saznanja o stanju celokupne mrežne infrastrukture, a ne pojedinih mrežnih segmenata. Drugim rečima, posmatrano sa aplikacionog sloja, cela mreža sa aspekta aplikacije deluje kao jedan veliki logički svič [18]. Može se zaključiti da su dve krucijalne karakteristike SDN koncepta:

- Velika programabilnost koja se manifestuje kroz:
  - Dinamičko upravljanje SDN mrežama korišćenjem softvera,
  - Primenu aplikativnih programskih interfejsa (API) otvorenog standarda, kako bi se mogle implementirati aplikacije za dinamičku i automatizovanu konfiguraciju i optimizaciju mrežnih resursa nezavisno od toga ko je proizvođač opreme.
- Apstrakcija koja omogućava da se:
  - Korisničke aplikacije izdvoje od mrežnih elemenata (mrežni uređaji su apstrahovani u odnosu na upravljački sloj),
  - Kontrolna ravan razdvoji od ravni prosleđivanja podataka i time stvore preduslovi za dinamičku alokaciju resursa prema potrebama korisnika,

- Vrši fleksibilna apstrakcija toka na osnovu *flow* tabele.

Upravljanje mrežnom infrastrukturom može se realizovati kroz *southbound* standardizovanog protokola poput *OpenFlow* protokola [19]. Ovaj protokol koristi koncept tokova (vrsta pravila protokola komunikacije) koji su definisani na samom SDN kontroleru. Pravila su grupisana u tabele tokova koje takođe kreira SDN kontroler. Pravila mogu biti statički definisana ili se mogu dinamički definisati u toku rada mreže. Na osnovu njih se programiraju, pa i prema njima ponašaju mrežni uređaji u SDN mreži. Na osnovu ovih tabela moguće je paralelno definisati ponašanje svih mrežnih uređaja i poboljšati njihove performanse, kao i značajno smanjiti kompleksnost upravljanja mrežnom infrastrukturom [20].

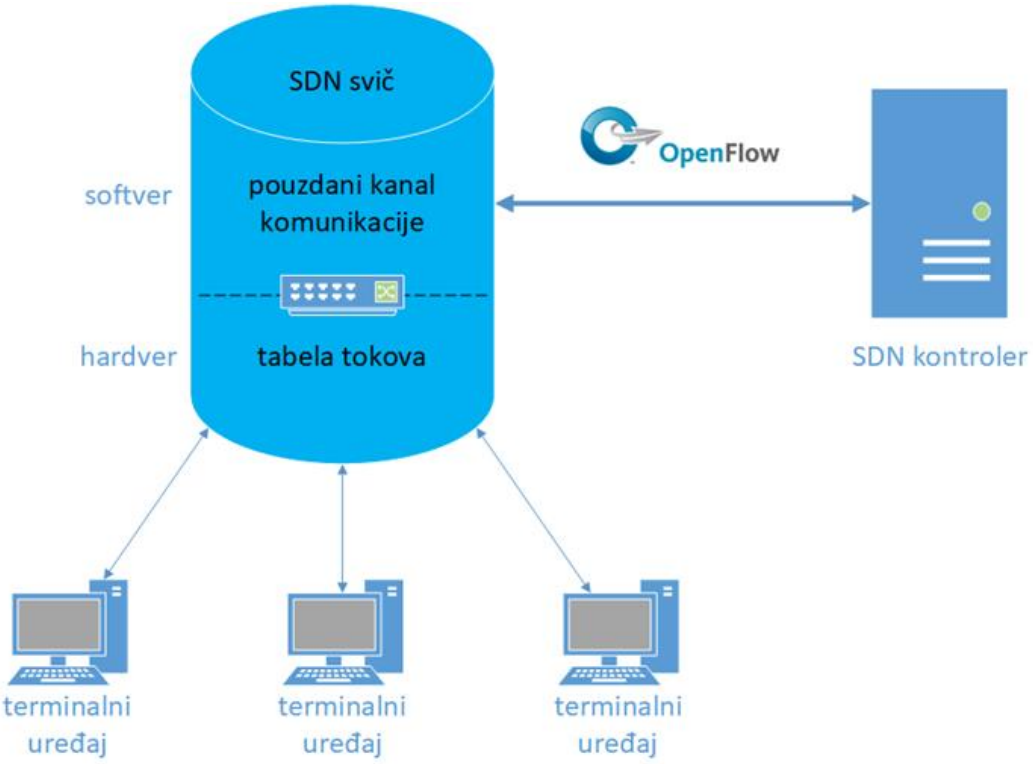
3.1 Teorijske osnove *OpenFlow* protokola

Jedan od protokola komunikacije u toku razvoja SDN tehnologije bio je *OpenFlow*. Protokol je predviđen za komunikaciju između SDN kontrolera i SDN sviča i on kao takav predstavlja srž *OpenFlow* tehnologije (slika 3.4). Protokol čini skup poruka koje se šalju od SDN kontrolera ka SDN sviču, kao i skup poruka koje se razmenjuju u suprotnom smeru. SDN kontroler na ovaj način direktno upravlja SDN svičem koji kao mrežni uređaj biva dinamički konfigurisan i prilagođen trenutnim potrebama u mreži (kad je u pitanju saobraćaj podataka).

Osnovni vid upravljanja svičem jeste kreiranje, izmena i brisanje instrukcija tokova u tabeli tokova na sviču. Tabela tokova je tabela sa postojećim instrukcijama prema kojim svič prosleđuje saobraćaj (pakete). Jedan skup instrukcija ukazuje na to koja pravila prosleđivanja svič treba da ispoštuje.

Kada kontroler pošalje sviču instrukciju, od sviča se očekuje da tu informaciju iskoristi za identifikaciju paketa čiji se deo sadržaja podudara sa podacima iz instrukcije, i preuzme odgovarajuće mere. Osnovna uputstva za tretiranje paketa od strane sviča jesu:

- Prosledi paket na jedan ili više izlaznih portova,
- Odbaci paket,
- Prosledi paket kontroleru na rukovanje izuzecima,
- Prosledi paket na sve portove osim ulaznog,
- Prosledi paket na sve portove osim ulaznog i svih onih koje STP (*Spannning Tree Protocol*) protokol ne predviđa i
- Obradi pakete služeći se tradicionalnim L2/L3 procesiranjem.



Slika 3.4 *OpenFlow* protokol

3.1.1 Tabela tokova

Tabela tokova je vrsta tabelarne strukture podataka koja u sebi sadrži definisana pravila prema kojim se prosleđuju paketi pristigli na svič. Smeštena je na SDN sviču, i na osnovu nje svič prepoznaje kojem entitetu mreže dalje prosleđuje pakete.

Komponente jednog toka u tabeli tokova date su slikom 3.5.

Polja podudaranja	Prioritet	Brojači	Instrukcije	Timeout	Kolačići

Slika 3.5 Komponente jednog unosa upravljačkog toka

Opisi komponenti unosa upravljačkog toka dati su u nastavku [21]:

- Polja podudaranja - pravila podudaranja unosa toka. Ona sadrže ulazni port, zaglavlja paketa i metapodatke;
- Prioritet - odgovarajući prioritet unosa toka. Kada se paket podudara sa tabelom toka, bira se samo unos toka najvišeg prioriteta koji odgovara paketu;
- Brojači - broj paketa koji odgovaraju unosu toka;

- Instrukcije - Koristi se za izmenu skupa radnji ili obrade *pipeline*-a;
- *Timeout* - maksimalno vreme mirovanja ili *hard time* za unos toka:
- vreme mirovanja - Unos toka se uklanja kada se nije uparío nijedan paket tokom vremena mirovanja;
- *hard time* - Unos toka se uklanja kada se prekorači vremensko ograničenje, bez obzira na to da li ima podudarnih paketa ili ne;
- Kolačić - identifikator unosa toka koji je odredio kontroler.

Tokovi mogu biti proaktivni ili reaktivni. Proaktivni tokovi su predefinisani tokovi, dakle njihovo prisustvo na sviču je stalno, nepromenljivo. Nalaze se na SDN sviču na samom početku rada mreže i nije predviđeno da se ikada uklanjaju iz tabele tokova.

Reaktivni tokovi nastaju kao posledica prijema određenog zaglavlja paketa, odnosno kao reakcija na pristigli paket. Oni nastaju i po mogućnosti nestaju iz tabele tokova u toku rada mreže.

SDN kontroler je zadužen za ubacivanje i brisanje i proaktivnih i reaktivnih tokova u tabeli lociranoj na SDN sviču.

Ono što može da utiče na redosled obrade paketa na sviču, jeste vrednost prioriteta tokova, definisana takođe u tabeli tokova. Paketi koji odgovaraju tokovima višeg prioriteta, obrađuju se pre paketa sa manjom vrednošću prioriteta toka.

3.1.2 Pregled SDN kontrolera koji koriste *OpenFlow* protokol

U Tabeli 3.1 [11][12][22] dat je pregled najpoznatijih *OpenFlow* SDN kontrolera koji su dostupni i javnosti. Nakon tabele sledi i pojedinačan kratak opis svakog od njih. Posebna pažnja posvećena je *NOX* kontroleru, kao originalnom SDN kontroleru koji je podržavao *OpenFlow* protokol komunikacije.

Naziv	Arhitektura	Jezik	Institucija	Verzija
Beacon	centralizovana, višenitna	Java	Stanford University	1.0
Floodlight	centralizovana, višenitna	Java	BigSwitch	1.0
Helios/Trema	centralizovana, višenitna	C/Ruby	NEC	1.0
IRIS	centralizovana, višenitna	Java	ETRI	1.0
Jaxon	Centralizovana	Java	-	1.0
NodeFlow	Distribuirana	JavaScript	-	1.0
NOX	Centralizovana	C++/Python	Stanford University	1.0
Opendaylight	Distribuirana	Java	Linux Foundation	1.0, 1.3
POX	Centralizovana	Python	Stanford University	1.0
Ryu	centralizovana, višenitna	Python	NTT	1.0, 1.2, 1.3

Tabela 3.1 Pregled *OpenFlow* SDN kontrolera

Tabela 3.1 sadrži osnovne informacije o najčešćim SDN kontrolerima u literaturi. Informacije koje sadrži su: naziv kontrolera, arhitektura softvera kontrolera, programski jezik u kome je softver kontrolera pisan, institucija iz koje potiče kontroler i verzija (verzije) *OpenFlow* protokola koju kontroler podržava.

Arhitektura softvera može biti centralizovana, višenitna, kao i distribuirana. Centralizovana arhitektura podrazumeva da je program kontrolera smešten na jednom centralnom mrežnom uređaju, odnosno na samom SDN kontroleru. Višenitna arhitektura podržava višenitno ili konkurentno programiranje (engl. *multi-threading*). Distribuirana arhitektura programa kontrolera znači da je deo tog programa i na drugim mrežnim uređajima.

Iz tabele se može videti da nekoliko kontrolera potiče sa Stanforda, univerziteta sa kojeg potiče i sam *OpenFlow* protokol. Kontroleri kod kojih je ova informacija u tabeli označena kao „-“ su proizvodi projekata individualaca, a ne neke institucije.

Kolona verzija odnosi se na verziju *OpenFlow* protokola, i jasno je iz tabele da svi navedeni kontroleri podržavaju makar prvobitnu verziju protokola (1.0).

NOX kontroler

NOX kontroler je prvobitni SDN kontroler sa *OpenFlow* protokolom. U radu [23] autori skreću pažnju na problem nedostatka „operativnog sistema” za mreže. Naime, baš poput računara koji kroz operativni sistem apstrahuju svoje fizičke resurse i time ih približavaju njihovim korisnicima, mrežama je takođe bio neophodan viši nivo za upravljanje i konfiguracije resursa. Upravljanje mrežnim resursima bilo je realizovano na dosta niskom nivou, konfiguracijom pojedinačnih

komponenti fizičke arhitekture. Uvođenjem tog tzv. operativnog sistema, upravlja se celom mrežom preko centralizovanog interfejsa i nudi se mogućnost nadgledanja mrežnih resursa. Uređaj na kome je bio implementiran taj operativni sistem (*NOX*) jeste SDN kontroler. Takođe, takav operativni sistem zahteva i pisanje programa (aplikacija) za upravljanje mrežnim resursima. Dakle, *NOX* ne nudi automatsko upravljanje mrežom sam za sebe, već je neophodno implementirati aplikaciju koja bi vršila tu funkciju.

Prvobitni *NOX* pisan je u C++ i *Python* programskim jezicima, a trenutna verzija je napisana u C++ jeziku. Iako sam *NOX* nije višenitni kontroler, postoji njegovo prošireno rešenje sa višenitnom arhitekturom (*NOX-MT*). Tokom proteklih godina, po uzoru na *NOX* napisani su i mnogi drugi kontroleri koji podržavaju *OpenFlow* protokol, a u nastavku će biti izdvojeni samo oni na koje se najčešće nailazi u literaturi.

Jaxon kontroler

Jaxon kontroler [24] je prilagodljiv SDN kontroler pisan u *Java* programskom jeziku. Zasnovan na *NOX* kontroleru, on u stvari predstavlja vezu između *Java*-e i *NOX* kontrolera.

Opendaylight kontroler

Opendalylight kontroler [25] je implementiran kao JVM (eng. *Java Virtual Machine*) programsko rešenje. Upotrebljiv je od strane svih operativnih sistema i fizičke arhitekture koji podržavaju *Java*-u. Pored *OpenFlow* protokola *OpenDaylight* je predviđen da radi i sa drugim protokolima komunikacije otvorenog standarda (primeri: I2RS, VxLAN, PCEP).

Kontroler raspolaže API-jima koje koriste aplikacije servisa. OSGi *framework* i dvosmerni REST su podržani za *northbound* API-je. OSGi framework se koristi za aplikacije koje se pokreću u istom adresnom prostoru kao i kontroler, dok se REST API koristi za aplikacije koje ne rade u istom adresnom prostoru (ili čak u istom sistemu) kao i kontroler. Poslovna logika i algoritmi se nalaze u aplikacijama. Ove aplikacije koriste kontroler za prikupljanje mrežne inteligencije, pokretanje njenog algoritma za obavljanje analitike, a zatim orkestriranje novih pravila širom mreže.

Na *southbound* strani, višestruki protokoli su podržani kao dodaci, npr. *OpenFlow* 1.0, *OpenFlow* 1.3, BGP-LS i tako dalje.

Floodlight kontroler

Floodlight kontroler [26] je prvobitno trebalo da pripadne *Opendaylight* projektu, ali se sticajem okolnosti ipak razvijao kao samostalan proizvod. Takođe je pisan u *Java* jeziku i podržava mnoge virtuelne i fizičke *OpenFlow* svičeve, ali i one svičeve koji ne podržavaju *OpenFlow* protokol.

Beacon kontroler

Beacon kontroler [27] radi brzo, podržan je od strane mnogih platformi (od *Linux* servera do *Android* pametnih telefona), modularan je i napisan u *Java*-i. Poseduje *web* interfejs, i izgrađen je iz više poznatih *Java framework*-a (*Spring, Equinox*).

IRIS kontroler

IRIS kontroler [28] je još jedan kontroler napisan u *Java*-i. Njegov kod je napisan po uzoru na *Floodlight* i *Beacon* kontrolere. Njegova optimizacija je fokusirana na poboljšanje skalabilnosti, pouzdanosti, ali i podrške više domena.

NodeFlow kontroler

NodeFlow kontroler [29] je *OpenFlow* kontroler sa minimalnim funkcionalnostima, napisan u *JavaScript* jeziku i predviđen za *Node.js*. *Node.js* je softverski sistem pogodan za pisanje proširivih *web* aplikacija, i koristi se na serverima.

Helios/Trema kontroler

Helios kontroler [30] je prvobitno bio napisan u C jeziku. Vremenom se proširio u *framework* za pisanje novih kontrolera i promenio naziv projekta u *Trema framework* [31]. *Trema framework* je predviđen za implementiranje SDN kontrolera pretežno u *Ruby* jeziku, iako postoji i podrška za C jezik. Biblioteke kojim su implementirane *OpenFlow* funkcionalnosti su na visokom nivou apstrakcije.

Ryu kontroler

Ryu kontroler [7] je pisan u *Python* jeziku. Njegov program je podeljen u komponente i vremenom se razvio u *framework* za umrežavanje. Pored *OpenFlow*-a podržava i druge protokole za upravljanje mrežama (primeri: *Netconf, OF-config*).

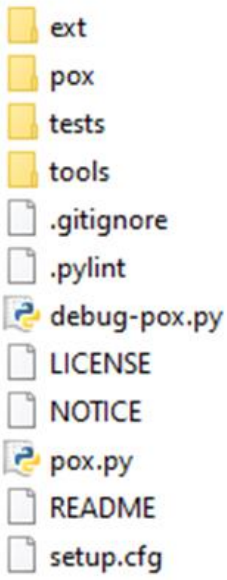
POX kontroler

POX kontroler je pisan u *Python* jeziku. Razvijan je na istom univerzitetu (Stanford) kao i njegov prethodnik *NOX*. *POX* će biti detaljnije opisan u nastavku teksta.

3.2 Dizajn *POX* kontrolera

POX kontroler [32] (u daljem tekstu *POX*) je SDN kontroler koji se kao i dosadašnji kontroleri zasniva na *OpenFlow* protokolu komunikacije. *POX* se može opisati kao rešenje za umrežavanje pisano u *Python* programskom jeziku. Poslednja verzija pisana za *Python 2* bila je *fangooth*, a počevši

od verzije gar *POX* zahteva *Python 3*. Gotovo sve platforme koje podržavaju *Python* mogu koristiti i *POX* softver, dakle, *Windows*, *Linux*, kao i *Mac OS*. Ipak, nisu sve funkcionalnosti podržane kod svih platformi, najveći broj funkcionalnosti je podržan od strane *Linux*-a. Za potrebe ovog rada bila je dovoljna *eel* verzija *POX* softvera (*Python 2*) i korišćen je kontroler sa *Linux* (*Ubuntu*) operativnim sistemom. Struktura *POX* projekta na *GitHub*-u data je slikom 3.6.



Slika 3.6 Organizaciona struktura projekta *POX*

Programsko rešenje *POX* pisano je uglavnom u skladu sa objektno-orijentisanim konceptima programiranja. To je jedno modularno rešenje, sastavljeno od raznih opcionih komponenti. Primena komponenti je jednostavna. Prilikom pokretanja glavne skripte u komandnoj liniji, *pox.py*, u nastavku se navedu sve komponente koje je potrebno pokrenuti. Primer:

```
./pox.py forwarding.l2_learning
```

U ovom slučaju pokrenuta je komponenta *l2_learning*, zadužena za prisluškivanje L2 saobraćaja. Komponente se uglavnom smeštaju u direktorijume samog projekta. Program ih pri pokretanju traži u *pox* i *ext* direktorijumima (slika 3.6), ali i na sistemskim putanjama. Svaka komponenta poseduje svoju *launch()* metodu, zaduženu za pokretanje te komponente, a koju poziva glavna skripta. Dakle, pokretanje softvera može se svesti na sledeće faze:

1. *pox.py* podigne *POX* program,
2. Pročita komponente prosleđene u komandnoj liniji,
3. Pronađe komponente u fajl sistemu,
4. Pozove *launch()* metode tih istih komponenti,
5. Program prelazi u režim rada.

POX nudi i korišćenje *debug-pox.py* skripte umesto glavne *pox.py* skripte, za potrebe uklanjanja grešaka u kodu tokom razvoja aplikacija.

3.2.1 *POX* komponente

Šta su *POX* komponente? Kao i drugi SDN kontroleri *POX* je takođe predviđen za upravljanje SDN mrežom iz aplikativnog sloja. Ovo podrazumeva implementiranje mrežnih aplikacija. Aplikacije se realizuju preko *POX* komponenti, a one najčešće podrazumevaju kreiranje i izmenu tabela tokova na SDN sviču po nekom unapred definisanom programu. Za razliku od nekih drugih kontrolera (na primer *Opendaylight*) *POX* kontroler ne koristi REST API ili slične interfejse, već koristi svoje komponente. Međutim, posredstvom ovih komponenti moguće je koristiti i neke drugačije dizajnirane API-je.

Dakle, *POX* komponente su dodatni *Python* programi koje *POX* pokrene pri pokretanju sopstvenog programa. One su sami nosioci mrežnih funkcionalnosti u SDN mrežama. Dok *POX* poseduje već implementirane komponente, on takođe ostavlja svom korisniku mogućnost za implementiranje novih.

Važno je istaći da *POX* (*pox.py* skripta) bez poziva bilo kakvih komponenti nema neku praktičnu funkcionalnost. Njegova svrha jeste da pokrene jednu ili više komponenti.

3.2.2 Komponenta za ispis poruka (*log.level*)

Iako je za potrebe ovog rada bila potrebna samo kreirana komponenta za balansiranje opterećenja servera, primenjena je i jedna pomoćna komponenta za potrebe kontrolisanog izvršavanja programa, a koja je sastavni deo *POX* softvera, to je *log.level* komponenta. Razne komponente poseduju sopstvene module za ispis poruka u radnoj konzoli nakon njihovog pokretanja. Svaka poruka ima svoj nivo, taj nivo odgovara važnosti odgovora softvera prilikom njegovog korišćenja. Lista ovih nivoa, od najvažnijeg do najmanje važnog jeste:

- CRITICAL
- ERROR
- WARNING
- INFO
- DEBUG

CRITICAL i ERROR poruke odgovaraju greškama koje se javljaju pri izvršavanju programkog rešenja *POX* kontrolera, dok su WARNING, INFO i DEBUG više informativnog sadržaja. Podrazumevani nivo pri pokretanju *log.level* komponente je INFO, dok je u ovom radu pretežno

korišćen DEBUG nivo, prilikom razvoja same aplikacije tehničkog dela rada. DEBUG nivo se koristi na sledeći način:

```
./pox.py log.level --DEBUG
```

3.2.3 *POX core* objekat

POX poseduje mnogo aplikativnih programskih sprega (API). Centralna tačka mnogih od tih sprega je *core* objekat. Najbitnija primena ovog objekta jeste obezbeđivanje skladne saradnje među komponentama. U *Python* jeziku uglavnom jedan modul uvozi drugi (*import* iskaz), te se na ovaj način, među modulima dobija odnos nadređen-podređen. Međutim, *core* objekat nudi mogućnost registrovanja (eng. *register*) komponenti i stoga, komponente mogu međusobno sarađivati slanjem upita *core* objektu. Glavna prednost ovakvog pristupa jeste to što međusobne zavisnosti komponenti nisu „zacementirane" u samom kodu, već se u skladu sa potrebama programa mogu jednostavno izmeniti. Na ovaj način se izbegavaju problemi koji mogu nastati rukovanjem *Python*-ovim prostorom imena. Sam *core* objekat se sa druge strane uvozi u komponente na standardni *Python* način:

```
from pox.core import core
```

Registrovati komponentu

Povoljno je da komponenta poseduje objekat registrovan na *core* objektu. Primer ove pogodnosti jeste često primenjivana instanca objekta *OpenFlow* komponente *core.openflow*, sa kojom su implementirane *OpenFlow* funkcionalnosti.

Postoje dva načina registrovanja novih komponenti, odnosno dve metode:

- *core.register()*
- *core.registerNew()*

core.register() metodi su potrebna dva argumenta pri pozivu, a to su korisnički definisan naziv komponente i objekat koji je potrebno registrovati na *core* objektu.

core.registerNew() prihvata samo jedan argument, a to je klasa kroz koju je komponenta implementirana.

Ove metode se pozivaju u okviru *launch()* metode.

Međusobna zavisnost i upravljanje događajima

Ako postoji više istovremeno registrovanih *POX* komponenti, to je najčešće zbog toga što je neophodno da jedna prisluškuje događaje druge. U tu svrhu *POX* poseduje prilično koristan mehanizam za njihovu međusobnu zavisnost - obe komponente zavise od još jedne (treće) komponente i to na isti način, a isti mehanizam takođe može upravljati događajima: *core.listen_to_dependencies(). listen_to_dependencies()* metoda ima nekoliko argumenata:

- *sink*
- *components* (podrazumevana vrednost: *None*)
- *attrs* (podrzumevana vrednost: *True*)
- *short_attrs* (podrazumevana vrednost: *False*)
- *listen_args* (podrazumevana vrednost je prazan rečnik)

U ovom radu su od interesa bili *sink* i *listen_args* argumenti. *sink* parametar je objekat iz kojeg se kreira lista komponenti preko *_handle_<ComponentName>_<EventName>*.

listen_args je rečnik koji nudi mogućnost definisanja dodatnih parametara.

3.2.4 Biblioteka za adrese (*pox.lib.addresses*)

U cilju izbegavanja problema sa formatiranjem vrednosti mrežnih adresa, *POX* ima implementirane i posebne klase za IP i *Ethernet* adrese:

• *IPAddr* – klasa za IPv4 adrese

• *IPAddr6* – klasa za IPv6 adrese

• *EthAddr* – klasa za *Ethernet* adrese

Primer korišćenja:

```
from pox.lib.addresses import IPAddr
ip = IPAddr("192.168.1.1")
```

Ova biblioteka poseduje i razne korisne funkcije, na primer: za parsiranje net-maski, CIDR (eng. *Classless Inter-Domain Routing*) notaciju, proveru pripadnosti IP adrese određenoj podmreži, itd.

3.2.5 Biblioteka za rukovanje događajima (*pox.lib.revent*)

Ova biblioteka služi za rukovanje događajima. Događaj u programiranju jeste trenutak na koji treba reagovati izvršavanjem određenog dela programa. Rukovanje događajima u *POX*-u je sprovedeno po sledećem uputstvu: Dok pojedini objekti pokreću događaje, drugi se prijavljuju za rukovanje konkretnim događajima pojedinih objekata.

Svi *POX*-ovi događaji su instance klase *revent.Event*. To je klasa koja je zadužena za pokretanje događaja, a ona takođe poseduje i listu onih događaja koje pokrene.

Postoji prečica za prisluškivanje višestrukih događaja istog izvornog objekta, a to je metoda: *addListeners().* Primera radi, funkcija *foo.addListeners(bar)* prvo pretražuje događaje objekta *foo*, a

zatim ako primeti da se metoda koja nosi naziv po konvenciji *_handle_*EventName** izvršava nad *bar* objektom, ona primenjuje tu metodu za prisluškivanje objekta *bar*.

Ova biblioteka takođe poseduje razne objekte, i ukoliko se oni primene kao povratne vrednosti, to rezultuje određenim akcijama. Na primer, *EventHalt* zaustavlja svaki drugi aktuelan događaj.

3.2.6 Biblioteka za pakete saobraćaja (*pox.lib.packet*)

Jedna od ključnih funkcionalnosti SDN kontrolera jeste mogućnost presretanja i analize paketa saobraćaja u cilju reagovanja na takve događaje. *POX* kontroler, pored mogućnosti kreiranja paketa, nudi i mogućnost primanja paketa u vidu *ofp_packet_in* poruka, koje odgovaraju *OpenFlow* protokolu. Preko *pox.lib.packet* biblioteke moguće je parsirati i kreirati pakete različitih tipova.

Paketi uglavnom poseduju zaglavlje i koristan sadržaj. Koristan sadržaj paketa često u sebi nosi i neki drugi tip paketa. Primera radi, *POX* radi sa *Ethernet* paketima, koji često sadrže IPv4 paketa, a IPv4 paketi sadrže TCP pakete. Neki od tipova paketa koje *POX* podržava su:

- ethernet
- ARP
- IPv4
- ICMP
- TCP
- UDP
- DHCP
- DNS
- LLDP
- VLAN

Biblioteka *pox.lib.packets*, odnosno njeni paketi se mogu koristiti klasičnim uvozom biblioteke u kod programa:

```
import pox.lib.packet as pkt
```

Jedan način da se parsiraju paketi jeste da se koristi *payload* atribut objekta paketa. Primer parsiranja paketa ICMP (eng. *Internet Control Messaging Protocol*) poruke:

```
def parse_icmp (eth_packet):


     if eth_packet.type == pkt.IP_TYPE:
       ip_packet = eth_packet.payload
       if ip_packet.protocol == pkt.ICMP_PROTOCOL:
      icmp_packet = ip_packet.payload
```

Na svakom nivou enkapsulacije (pakovanja) paketa, mogu se dobaviti vrednosti zaglavlja paketa.

Primer dobavljanja IP adrese izvora IP paketa:

```
src_ip = ip_packet.srcip
```

Konstruktor paketa

Objekat paketa potrebno je konvertovati iz objekta u niz bajtova i pri tome se može koristiti *pack()* metoda nad tim objektom. Kada je potrebno konvertovati niz bajtova u objekat koristi se metoda *unpack()*. Pri radu sa *PacketIn* objektima, konverzija iz niza bajtova u objekat je automatizovana. *PacketIn* objekti će biti ponovo pomenuti kasnije, oni u principu sadrže atribut: *packet*, u kojem je sadržan objekat paketa. Postoje slučajevi u kojima je neophodno ručno raspakovati seriju bajtova kako bi se dobio odgovarajući objekat paketa, ali u ovom radu to nije bilo neophodno.

Ovaj rad koristi metodu *pack()* i to u funkciji pakovanja podataka u *Ethernet* paket, a prilikom slanja paketa odgovora od kontrolera ka sviču.

3.2.7 „Obične" niti u okviru *POX* komponenti

Iako *POX* poseduje sopstvenu biblioteku (*pox.lib.recoco*) za rukovanje nitima za ciljeve konkurentnog programiranja, podržana je i primena standardnog *Python* modula: *threading*. Taj modul je i primenjen za potrebe ovog rada, i to kod kontinualnog dobavljanja SNMP podataka o resursima server mašina.

Problem sa normalnim nitima može nastati samo ukoliko je potrebno sinhronizovati *recoco* nit sa normalnom niti koja potiče od *threading* modula.

3.2.8 Veza kontrolera i sviča

Svaki put kada se svič poveže sa *POX* kontrolerom, nastane objekat *Connection*. Ovaj objekat je moguće koristiti za slanje poruka sviču metodom *send()*.

Događaj *ConnectionUp* izvršava se na SDN kontroleru čim se uspostavi veza sa svičem kojim se kontroliše i to je prvi signal da je veza uspostavljena (odnosno da postoji objekat *Connection*). *ConnectionUp* poseduje atribut: *ofp_switch_features*, a ovaj atribut sadrži razne informacije o sviču (tipove dozvoljenih akcija, MAC adresu, itd.)

Događaji koji podrazumevaju *OpenFlow* protokol uglavnom imaju veze sa direktnim odgovorom kontrolera sviču. Ovi događaji imaju sledeća tri atributa, prikazana u tabeli 3.2.

atribut	tip podatka	opis
connection	Connection	veza ka sviču koji je poslao poruku događaja
dpid	Long	Datapath ID sviča koji je poslao poruku događaja
ofp	ofp_header podklasa	OpenFlow poruka koja je izazvala događaj

Tabela 3.2 Atributi *OpenFlow* događaja

Postoji i *ConnectionDown* događaj koji se izvršava kada se veza sa svičem ukine.

PacketIn događaj se izvršava kada kontroler primi od sviča *OpenFlow* paket (*ofp_packet_in*/OFPT_PACKET_IN), ovaj događaj ukazuje na to da paket pristigao na svičev port, ili ne odgovara nijednom aktuelnom pravilu iz tabele tokova, ili je upravo pravilo iz tabele tokova rezultovalo da se paket pošalje kontroleru.

U tabeli 3.3 prikazani su dodatni atributi *PacketIn* događaja.

atribut	tip podatka	opis
port	Int	port na kojem je paket ušao
data	Byte	neobrađen sadržaj paketa
parsed	packet podklasa	parsirana verzija paketa biblioteke pox.lib.packet
ofp	ofp_packet_in	OpenFlow poruka koja je izazvala događaj

Tabela 3.3 Dodatni atributi *PacketIn* događaja

3.2.9 *OpenFlow* poruke

Svičevi i kontroleri međusobno komuniciraju preko *OpenFlow* poruka. Postoji više verzija *OpenFlow* protokola, a *POX* kontroler podržava samo 1.0.0 verziju protokola.

Postoji biblioteka *pox.openflow.libopenflow_01* (gde je 01 oznaka verzije protokola) i ona poseduje klase i konstante koje predstavljaju odgovarajuće elemente *OpenFlow* protokola. Korisna strana ove biblioteke je što mnoga polja poruka imaju značajne podrazumevane vrednosti ili se iz njih mogu izvesti potrebni podaci. Slede opisi bitnijih sprega ove biblioteke, koji su između ostalog bili potrebni i prilikom izrade tehničkog dela ovog rada.

Slanje paketa od sviča (*of_packet_out*)

Na osnovu *of_packet_out* poruke svič šalje paket ili ga postavlja u red čekanja za slanje. Sa ovom porukom, takođe je moguće uputiti svič na odbacivanje poruke. Poruku opisuju atributi u sledećoj tabeli:

atribut	tip podatka	opis
buffer_id	int/None	ID buffer-a u kojem je paket smešten na datoj putanji. Ako se buffer ne šalje (ponovljeno slanje) po ID-ju, postavlja se na None.
in_port	Int	Sviček port na koji je paket pristigao u slučaju ponovljenog slanja.
actions	lista ofp_action_XXXX	U slučaju samo jedne akcije, može se koristiti parametar "action" pri inicijalizaciji.
data	bytes/ethernet /ofp_packet_in	Podaci koji se šalju (ili *None* ukoliko se šalje postojeći buffer preko svoje buffer_id vrednosti). U slučaju korišćenja ofp_packet_in podatka, in_port, buffer_id, i data atributi će svi biti postavljeni na ispravne vrednosti.

Tabela 3.4 Atributi *ofp_packet_out* objekta

Izmene tabele toka (*of_flow_mod*)

Poruka *of_mod_flow* ima mnogo parametara i atributa, ovde će biti izloženi samo oni koji su primenjeni u izradi tehničkog rešenja rada.

Od parametara bi trebalo izdvojiti *command* parametar koji može imati jednu od sledećih vrednosti:

- OFPFC_ADD – podrazumevana vrednost koja dodaje pravilo
- OFPFC_MODIFY – modifikuje pravila kod kojih se bilo šta od parametara pravila (poruke) podudara
- OFPFC_MODIFY_STRICT – modifikuje samo pravila koja odgovaraju „*wildcard*" vrednostima
- OFPFC_DELETE - briše pravila kod kojih se bilo šta od parametara pravila (poruke) podudara
- OFPFC_DELETE_STRICT – briše samo pravila koja odgovaraju „*wildcard*" vrednostima.

Tabela primenjenih atributa data je tabelom 3.5.

atribut	tip podatka	opis
priority	Int	U slučaju podudaranja paketa sa više pravila, biće uzeto u obzir pravilo sa najvećom vrednošću prioriteta.
actions	Lista	Lista akcija. Akcije su opisane u nastavku teksta. Svaka akcija se redom ubacuje u ovu listu i potom se akcije izvršavaju u istom redosledu u kojem su dospele u listu.
match	ofp_match	Struktura pravila podudaranja paketa. Opisano u nastavku teksta.

Tabela 3.5 Atributi *of_mod_flow* objekta

3.2.10 Strukture pravila podudaranja paketa

Zaglavlja paketa sadrže meta podatke, poput IP adresa, tipova paketa itd. Objekat *ofp_match* je struktura kojom *OpenFlow* definiše podudaranje zaglavlja paketa sa odgovarajućim skupom pravila vezanih za prisustvo meta podataka u zaglavljima paketa. *POX* omogućava kreiranje ovakve strukture iznova, ali i primenu metode za kreiranje strukture na osnovu postojećih paketa.

Definicije podudaranja paketa implementirane su već pomenutom bibliotekom *pox.openflow.libopenflow_01* i to kroz klasu *ofp_match*. Atributi ove klase su izvedeni iz *OpenFlow* dokumentacije, a ovde su izloženi narednom listom:

- *in_port* - Sviječev port na koji je paket pristigao
- *dl_src* - Adresa izvora *Ethernet* paketa
- *dl_dst* - Adresa destinacije *Ethernet* paketa
- *dl_vlan* - ID VLAN-a
- *dl_vlan_pcp* - Prioritet VLAN-a
- *dl_type* - Tip paketa/dužina (na primer: 0x0800 = IPv4)
- *nw_tos* - IP TOS/DS biti
- *nw_proto* - IP protokol (na primer, 6 = TCP) ili prvih 8 bita u slučaju ARP-a
- *nw_src* - Adresa izvora IP paketa
- *nw_dst* - Adresa destinacije IP paketa
- *tp_src* - Port izvora TCP/UDP paketa
- *tp_dst* - Port destinacije TCP/UDP paketa

Postoje dva načina da se atributi nove strukture podudaranja inicijalizuju. Prvi način da se inicijalizuju jeste pri kreiranju samog objekta klase *ofp_match*, na primer:

```
my_match = of.ofp_match(in_port = 5,
                        dl_dst = EthAddr("01:02:03:04:05:06"))
```

Drugi način da se atributi strukture podudaranja inicijalizuju jeste da se to učini nakon kreiranja objekta klase *ofp_match*, odnosno na već inicijalizovanom objektu strukture:

```
my_match = of.ofp_match()
my_match.in_port = 5
my_match.dl_dst = EthAddr("01:02:03:04:05:06")
```

Ova dva načina inicijalizacije atributa su ekvivalentna.

3.2.11 Akcije *OpenFlow* protokola

Pod akcijama se podrazumeva niz radnji koje svič treba da preduzme kada mu kontroler to saopšti. One se vezuju za trenutak kada dođe do podudaranja paketa sa prethodno objašnjenim pravilima podudaranja paketa.

U ovom radu su primenjivane tri vrste akcija, odnosno metoda, a to su:

- Izlaz (*Output*),
- Postavljanje *Ethernet* adrese izvora ili destinacije paketa i
- Postavljane IP adrese izvora ili destinacije paketa.

Akcija izlaza *ofp_action_output* prosleđuje pakete na fizički ili virtuelni port sviča. Fizički portovi su označeni celobrojnim vrednostima, a virtuelni portovi poseduju simbolična imena. Fizički portovi moraju posedovati vrednosti ispod heksadecimalne vrednosti 0XFF00. Simbolična imena portova mogu biti sledeća:

- OFPP_IN_PORT – paket se šalje na isti port na koji je i pristigao
- OFPP_TABLE – paket se preusmerava po uputstvima iz tabele tokova (primenjivo samo na *ofp_packet_out* porukama)
- OFPP_NORMAL – paket se šalje prema tradicionalnoj L2/L3 svič konfiguraciji
- OFPP_FLOOD – izlaz paketa se obezbeđuje na svim *OpenFlow* portovima sviča izuzev ulaznog i onih kod kojih je onemogućen *flooding* mehanizam preko OFPPC_NO_FLOOD konfiguracionog bita
- OFPP_ALL - izlaz paketa se obezbeđuje na svim *OpenFlow* portovima sviča izuzev ulaznog
- OFPP_CONTROLLER – paket se šalje kontroleru
- OFPP_LOCAL – paket se šalje lokalnom *OpenFlow* portu
- OFPP_NONE – paket se ne šalje

Akcija postavljanja *Ethernet* adrese izvora ili destinacije paketa (*ofp_action_dl_addr*) se koristi pri postavljanju nove izvorne ili destinacione *Ethernet* adrese paketa od interesa. Postoje dva parametra koje klasa ove akcije prihvata pri kreiranju svog objekta, a to su tip i vrednost same MAC (*Ethernet*) adrese koju je potrebno postaviti. Tip može imati vrednosti:

- OFPAT_SET_DL_SRC – ukoliko je potrebno postaviti MAC adresu izvora
- OFPAT_SET_DL_DST – ukoliko je potrebno postaviti MAC adresu destinacije

Akcija postavljanja IP adrese izvora ili destinacije paketa (*ofp_action_nw_addr*) koristi se pri postavljanju nove izvorne ili destinacione IP adrese paketa od interesa. Postoje dva parametra koje klasa ove akcije prihvata pri kreiranju svog objekta, a to su tip i vrednost same IP adrese koju je potrebno postaviti. Tip može imati vrednosti:

- OFPAT_SET_NW_SRC – ukoliko je potrebno postaviti IP adresu izvora
- OFPAT_SET_NW_DST – ukoliko je potrebno postaviti IP adresu destinacije

3.3 Modeli implementacije SDN funkcionalnosti u računarskim mrežama

SDN mreže su, kao koncept, odavno prisutne u mnogim modernim računarskim, ali i industrijskim mrežama, na primer: u centrima za skladištenje podataka, u mrežama u preduzećima i kompanijama, u fabrikama za proizvodnju koje koriste *Industry* 4.0 tehnologiju i u 5G, 6G mobilnim mrežama. Efikasnost implementacije takvih mreža postiže se primenom različitih tipova virtuelizacija u mreži, a samim time i virtuelnih mrežnih funkcija (eng. *Network Function Virtualizatio*n, NFV). Ovi procesi virtuelizacije omogućavaju jednostavnije upravljanje (virtuelizovanim) resursima, jer nije potrebno zalaziti u detalje njihove hardverske realizacije, već se to obavlja nad apstrahovanom fizičkom arhitekturom [33]. Takođe, moguće je jednostavnije distribuirati konfiguraciju SDN mreže na nove mrežne resurse, te se povećava i skalabilnost celog sistema, naročito u centrima za skladištenje velike količine podataka. Pošto se virtuelne mašine jednostavnije prenose u nove mreže (sisteme) to pozitivno utiče i na upotrebljivost postojećih rešenja [34]. SDN rešenje primenjuje precizno definisanu apstrakciju infrastrukturnog sloja, koje uključuje modele apstrakcije prosleđivanja paketa, modele apstrakcije prebacivanja kola, integraciju bežičnih tehnologija, i napredne podrške paketa podataka.

Virtuelizacijom mrežnih funkcionalnosti, tradicionalne mrežne funkcije poput *firewall*-a i balansiranja saobraćaja premeštaju se što je moguće bliže željenoj lokaciji u mreži i sadržane su u samim aplikacijama. Ovim se postiže primena više logičkih mreža na zajedničkoj fizičkoj infrastrukturi. U nekim istraživanjima, predlaže se dodela resursa za *Industry* 4.0 tehnologiju,

zasnovanu na SDN i NFV tehnologijama, alatima za mašinsko učenje i podelu mreže na osnovu zahteva servisa (propusni opseg, kašnjenje, pouzdanost) [35]. Pored podele mrežnih resursa u cilju optimizacije, često se predlaže i promena na polju arhitekture mreže, kako bi se poboljšala i moć obrade podataka, ali i zaštita podataka po pitanju privatnosti [36].

3.4 Koncept hibridne SDN mreže

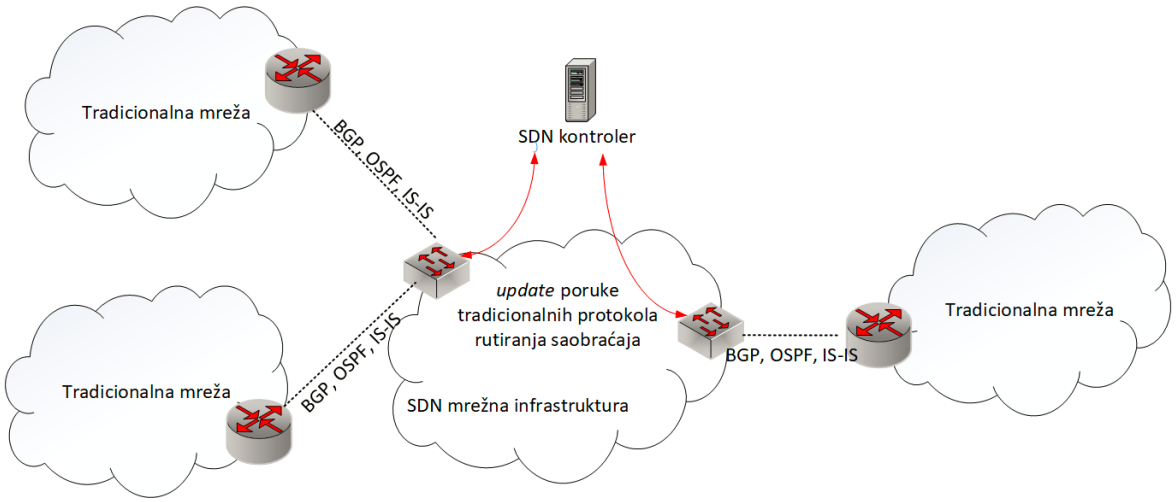
Potpuni prelazak sa tradicionalne na SDN mrežu, zahteva zamenu tradicionalnih svičeva – svičevima čije se konfiguracije mogu menjati prema uputstvima (instrukcijama) SDN kontrolera. Ove instrukcije se prenose *OpenFlow* porukama. Uz prelazak na SDN mrežu, ovim se postiže i uvođenje same SDN funkcionalnosti u mrežu. Zamena, tradicionalnih sa SDN svičevima, predstavlja ozbiljan izazov i finansijski zadatak u procesu sprovođenja i realizacije SDN koncepta u realnim sistemima. Razlozi su sledeći:

1. Potrebno je obzebediti značajna finansijska sredstva za zamenu velike količine postojeće mrežne opreme novom opremom.
2. Mogući su prekidi rada mreže ili njenih delova usled izvođenja potrebnih radova na mreži.
3. Sve postojeće hardverske mrežne funkcije treba implementirati u softveru.

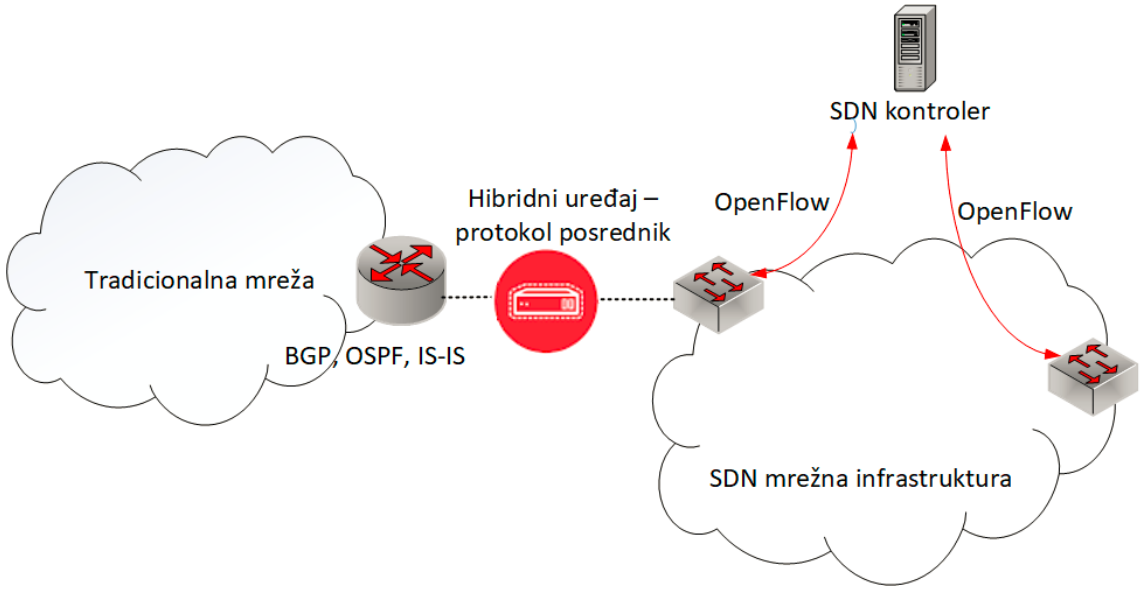
Imajući u vidu, prethodno pomenute razloge, većina mrežnih operatora se ipak opredeljuje za delimičnu tj. inkrementalnu primenu SDN funkcionalnosti i to kroz realizaciju koncepta hibridne SDN mreže. Umesto zamene svičeva u celoj mreži, to se čini samo nad pojednim njenim delovima. Na taj način, samo svičevi za čijim programiranjem postoji realna potreba bivaju zamenjeni svičevima koji su svesni prisustva SDN kontrolera.

Postoje dva tipa SDN sviča [37]:

a) SDN svič koji pored SDN funkcionalnosti prepoznaje i tradicionalne funkcionalnosti sviča (slika 3.7)
b) SDN svič koji ne prepoznaje tradicionalne funkcionalnosti sviča (slika 3.8)



Slika 3.7 SDN svič koji pored SDN funkcionalnosti prepoznaje i tradicionalne funkcionalnosti



Slika 3.8 SDN svič koji ne prepoznaje tradicionalne funkcionalnosti

Tip a) je potpuniji kada su funkcionalnosti u pitanju. Pored njegove programabilnosti, svič može da poseduje i sve funkcionalnosti običnog, tradicionalnog sviča, što oslobađa SDN kontroler od dužnosti upravljanja tradicionalnim protokolima komunikacije u mreži. U slučaju tipa b) kompletan protokol komunikacije za koji je svič zadužen realizovan je kroz program SDN kontrolera. Posledica toga jeste složeniji program za upravljanje mrežom, odnosno potreba za realizacijom složenijeg protokola komunikacije.

U određenim situacijama potrebno je povezati jednu ili više SDN mreža sa tradicionalnim mrežnim uređajima, i koncept hibridne SDN mreže je u tom slučaju neizbežan [38].

Drugim rečima, implementacija SDN mreža nosi sa sobom veliki broj pre svega tehničkih, ali i finansijskih i poslovnih izazova. S obzirom na te izazove, implementacija SDN funkcionalnosti u računarskim mrežama predstavlja veoma kompleksan posao, jer zahteva dobro osmišljenu strategiju zamene/implementacije opreme nad celom mrežom ili njenim segmentom, kao i vreme. Najveći broj mrežnih operatora, pogotovo onih velikih, upravo iz prethodno navedenih razloga, ima problem da u potpunosti implementira SDN koncept. Zato se oni uglavnom opredeljuju, za inkrementalnu primenu SDN funkcionalnosti i to kroz zamenu dela fizičkih uređaja sa virtuelnom infrastrukturom koja podržava rad sa *OpenFlow* protokolom. U praksi se mogu naći i uređaji na kojima je moguće implementirati *OpenFlow* protokol. Dakle, inkrementalnom implementacijom SDN funkcionalnosti dobija se hibridna arhitektura u kojoj SDN kontroler sa postojećom (tradicionalnom) infrastrukturom komunicira preko BGP ili OSPF protokola, a sa SDN infrastrukturom preko *OpenFlow* protokola i na taj način gradi svojevrsnu *overlay* mrežu.

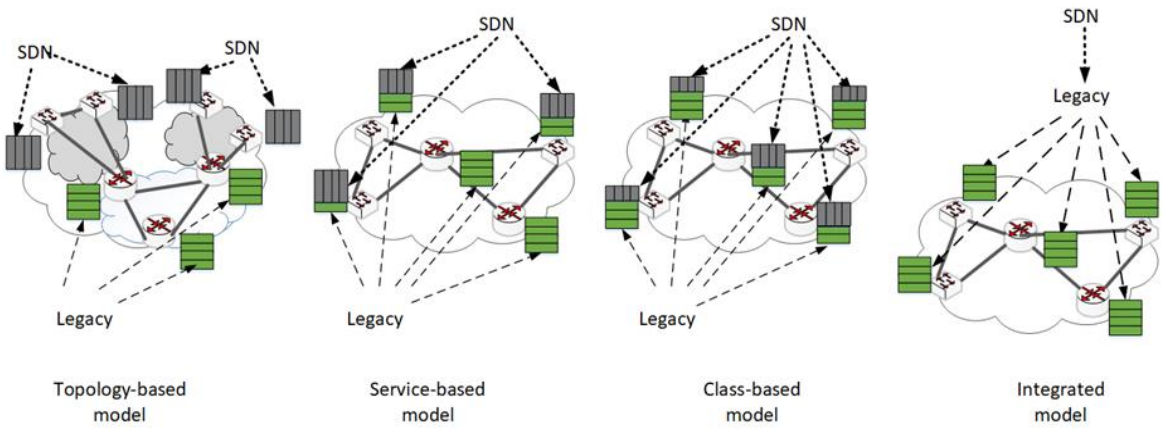
Hibridna SDN mreža se nameće kao prirodno rešenje, koje predstavlja kompromis između tehničkih potreba korisnika s jedne strane i finansijskih mogućnosti vremena potrebnog za realizaciju s druge strane. Zato je neophodno još jednom napomenuti, da uvođenje SDN koncepta i podrazumeva postepenu zamenu tradicionalne mrežne opreme sa SDN uređajima i potrebu da se jedan ovakav proces uspešno sprovede kroz precizno definisanu strategiju postepenog razvoja i implementacije SDN funkcionalnosti [39].

Kod hibridnih SDN mreža postoje i centralizovane i decentralizovane arhitekture, koje moraju međusobno komunicirati, kako bi kvalitet iskustva korisnika bio optimizovan kroz upravljanje i konfiguraciju mreže.

Tri ključne osobine, koje karakterišu hibridne SDN mreže jesu:

a) koegzistencija – heterogenost infrastrukture u sloju infrastrukture ili i sloju infrastrukture i sloju upravljanja;
b) komunikacija – interakcija tradicionalne opreme i SDN uređaja u cilju međusobnog sporazumevanja, i podele funkcionalnosti u sloju infrastrukture, sloju upravljanja ili i sloju infrastrukture i sloju upravljanja;
c) ukrštanje – kombinacija različitih mrežnih paradigmi sa komplementarnim karakteristikama kako bi se minimizovao budžet, pojednostavili prelasci na SDN infrastrukturu, automatizacija, i pravila saobraćanja podataka i postigla visoka skalabilnost i robusnost.

Posmatrano u celini, moguće je izvršiti klasifikaciju hibridnih SDN mreža na četiri modela [40], kao što je ilustrovano na slici 3.9:



Slika 3.9 Četiri modela hibridne SDN mreže prema klasifikaciji u [40]

Prvi model zasnovan je na topologiji mreže i podrazumeva podelu mreže na zone. Svaki mrežni uređaj može biti član samo jedne zone (tradicionalni mrežni uređaji ili SDN uređaji), dok samu zonu čini više uređaja kontrolisanih od strane iste mrežne paradigme. Drugi model je zasnovan na mrežnim servisima i potrebno je imati u vidu da tradicionalni i SDN mrežni uređaji snabdevaju različite servise. Takođe, potrebno je naglasiti, da kod ovog modela, *End-to-end* servisi mogu zahtevati da dve mrežne paradigme istovremeno upravljaju grupom uređaja, a da istovremeno neki uređaji mogu biti kontrolisani od strane samo jedne mrežne paradigme. Treći model je zasnovan na klasama, što podrazumeva podelu mrežnog saobraćaja na klase koje mogu biti kontrolisane od strane tradicionalnih mrežnih paradigmi ili SDN paradigmi. U ovakvom modelu uobičajeno je da svi uređaji imaju i tradicionalne i SDN funkcionalnosti. Četvrti model hibridne SDN mreže je integrisani hibridni SDN model. Kod ovog modela, SDN je odgovoran za sve mrežne servise. On koristi tradicionalne mrežne protokole kao spregu sa bazama za prosleđivanje informacija. SDN kontroler upravlja tradicionalnom paradigmom i bazom za prosleđivanje informacija.

Važno je napomenuti, da mrežna infrastruktura budućih 5G i 6G mreža mora biti koncipirana kao hibridna SDN mreža. Razlog za to leži u činjenici, da sve mrežne entitete i odgovarajuća tehničko-tehnološka rešenja vezana za prethodne generacije mobilnih mreža (2G, 3G i 4G) nije moguće istovremeno zameniti, pa je neophodno obezbediti podršku za njihovo normalno funkcionisanje i integrisati ih u 5G/6G eko sistem.

4. Razvoj modela hibridne SDN mreže

Cilj implementacije hibridne SDN arhitekture jeste, da se postigne optimalan rad mreže i to pre svega u pogledu mogućnosti njenog jednostavnog proširenja i povećanja pouzdanosti u radu. Zato je neophodno primeniti optimalnu logiku za upravljanje mrežnim saobraćajem. Da bi se ista logika uopšte mogla primeniti, potrebno je obezbediti pravovremeno i efikasno pribavljanje informacija kao što je opterećenost resursa na serverima. Kao osnovni zahtev u realizaciji hibridne SDN mreže, nameće se potreba postepenog uvođenja SDN funkcionalnosti primenom minimalnog broja SDN komponenti (na primer SDN svič i SDN kontroler). Na ovaj način, SDN funkcionalnost se uvodi u mrežu uz minimalne troškove, čime se postiže znatno veći stepen programabilnosti i stvaraju preduslovi za primenu naprednijih rešenja za upravljanje saobraćajem u mreži (npr. primenom algoritama za efikasnije balansiranje opterećenja u mreži).

Iako rešenje predstavljeno ovim istraživanjem sadrži minimalan broj uređaja, ono se može primeniti i u bilo kojem realnom okruženju. Ovde je veoma važno napomenuti, da je u slučaju primene rešenja u realnom mrežnom okruženju, potrebno obezbediti redundansu SDN komponenti kako bi se garantovala otpornost sistema na otkazivanje.

4.1 Pregled postojećih modela hibridne SDN mreže

Infrastruktura hibridne SDN mreže ima svoja ograničenja [39]. Jedno od tih ograničenja jeste kompleksno regulisanje procesa na sloju upravljanja, gde različite promene konfiguracije mreže mogu uticati na nekonzistentno ponašanje rutera [40] i dovesti do stvaranja petlji u prosleđivanju paketa podataka, ali i do tzv. crnih rupa u saobraćaju (mesto u mreži gde se poruke odbacuju bez obaveštavanja njihovog izvora) [41].

Još jedan značajan izazov u funkcionisanju hibridnih SDN mreža, svakako predstavlja i potreba međusobnih „prevođenja" (konverzija) iz tradicionalnih protokola komunikacije u SDN protokole, i obrnuto. To dovodi do pogoršavanja performansi, po pitanjima kašnjenja i procesorskog vremena. Ukoliko se koristi više od jednog SDN kontrolera, u mreži može nastati kašnjenje, ali i razni drugi problemi (regulisanje saobraćaja, proširivost i bezbednost). Regulisanje saobraćaja može postati vrlo izazovno ako svaki uređaj ne poseduje (ne podržava) apstrakciju tokova podataka [42]. Ipak hibridne mreže mogu biti adekvatno rešenje za postizanje željenog kvaliteta servisa u konvergentnim mrežama [43].

Razvoj hibridne SDN mreže uključuje zamenu tradicionalne opreme SDN uređajima. U praksi postoji *Panopticon* pristup [44], kojim se hibridna SDN mreža kreira povezivanjem tradicionalne i SDN mrežne opreme, tako da se sav saobraćaj usmerava na SDN svič kontrolisan SDN kontrolerom. Iako je ovim postupkom došlo do poboljšanja performansi mreže u odnosu na prvobitnu arhitekturu mreže, neki od važnih parametara mreže nisu uzimani u razmatranje (kašnjenje i uravnoteženje saobraćaja).

Ograničenja hibridnih SDN mreža, definisana u [40] i [41] prevazilaze se primenom programa SDN kontrolera sa precizno definisanim pravilima automatske konfiguracije (i rekonfiguracije) SDN mrežnih uređaja (rutera i svičeva), što ovo istraživanje i nastoji da pokaže. Prevazilaženje nedovoljno visoke apstrakcije tokova podataka na SDN uređajima takođe je predmet ovog istraživanja, kako bi se izbegli problemi opisani u [42]. Ovo je učinjeno primenom virtualnih IP adresa, na mestima gde je to neophodno, dok je sama uloga konverzije virtuelne u realnu IP adresu i obrnuto pripala SDN kontroleru.

Umesto usmeravanja celokupnog saobraćaja na SDN svič kao u [44], SDN sviču se samo prosleđuje saobraćaj neophodan za predviđen rad mreže. Kašnjenje i uravnoteženje saobraćaja takođe je adresirano primenom i testiranjem predloženog mehanizma (algoritma), opisanog kasnije u tekstu.

Dakle, jasno je, da bi provajderi internet servisa, postepenom primenom SDN funkcionalnosti mogli da iskoriste prednosti SDN tehnologije. Konkretno, postepenom implementacijom SDN opreme u mrežu, moguće je uneti veći nivo programabilnosti i iskoristiti prednosti takve mreže kroz upravljanje saobraćajem na željeni i što ne reći fleksibilniji način [45]. Provajderi mogu klasifikovati sam saobraćaj na osnovu toga da li on zahteva prisustvo, tj. implementaciju SDN sviča i to na programabilan i neprogramabilan deo mreže. U nekim slučajevima, čak nije ni neophodno da se zameni postojeća oprema, već je dovoljno samo implementirati novu SDN opremu [46], posebno imajući u vidu da SDN uređaji ne zahtevaju da se iz postojeće mreže isključe tradicionalni protokoli komunikacije. Drugim rečima, SDN kontroler će obrađivati samo saobraćaj koji je njemu od značaja.

U nekim istraživanjima [47], razvijena je ideja o primeni novih hibridnih SDN uređaja, u vidu dodataka tradicionalnoj mrežnoj opremi. Ovde je potrebno istaći, da su hibridni uređaji svesni

postojanja SDN kontrolera (njegovih instrukcija) u mreži. Ovakva implementacija je vredna pažnje, pogotovo u slučajevima, kada se razmatra bežični hibridni SDN model zasnovan na servisima [48]. Ovaj model se pokazao efikasnim, jer daje poboljšane performanse kad je u pitanju otpornost na greške u putanjama paketa podataka.

U praksi se hibridne SDN mreže mogu koristiti i za automobilske *ad-hoc* mreže, pri čemu je omogućena *End-to-end* razmena podataka sa fleksibilnim, skalabilnim i programabilnim funkcionalnostima. Ovakve mreže bi mogle da reše problem tradicionalnog upravljanja saobraćajem sa promenljivim uslovima u mreži [49]. Za zahtevne obrade podataka, istraživači se često bave naprednom analitikom podataka [50] i to još dok su podaci u fazi prenosa u hibridnoj SDN mreži. Kako bi se ostvarile napredne obrade podataka i izbegla eventualna kašnjenja u mreži nastala usled tih obrada, u radu [50] se predlaže primena FPGA kola (eng. *Field Programmable Gate Array*) i GPU kartica (eng. *Graphics Processing Unit*).

4.2 Metodologija balansiranja saobraćaja u hibridnim SDN mrežama

Na slici 4.1 prikazana je uprošćena arhitektura hibridne SDN mreže. Svič koji povezuje server mašine sa ostatkom mreže zamenjen je SDN svičem. Ovim postupkom je primenjena parcijalna implementacija SDN mreže i kreirana hibridna SDN mreža. Pored uobičajenih, tradicionalnih protokola komunikacije, kreirana su dodatna pravila za prosleđivanje saobraćaja na SDN sviču. Pomenuta pravila se skladište na SDN sviču, posredstvom programa (softvera) SDN kontrolera. Pravila se prosleđuju SDN sviču od strane SDN kontrolera primenom *OpenFlow* protokola.

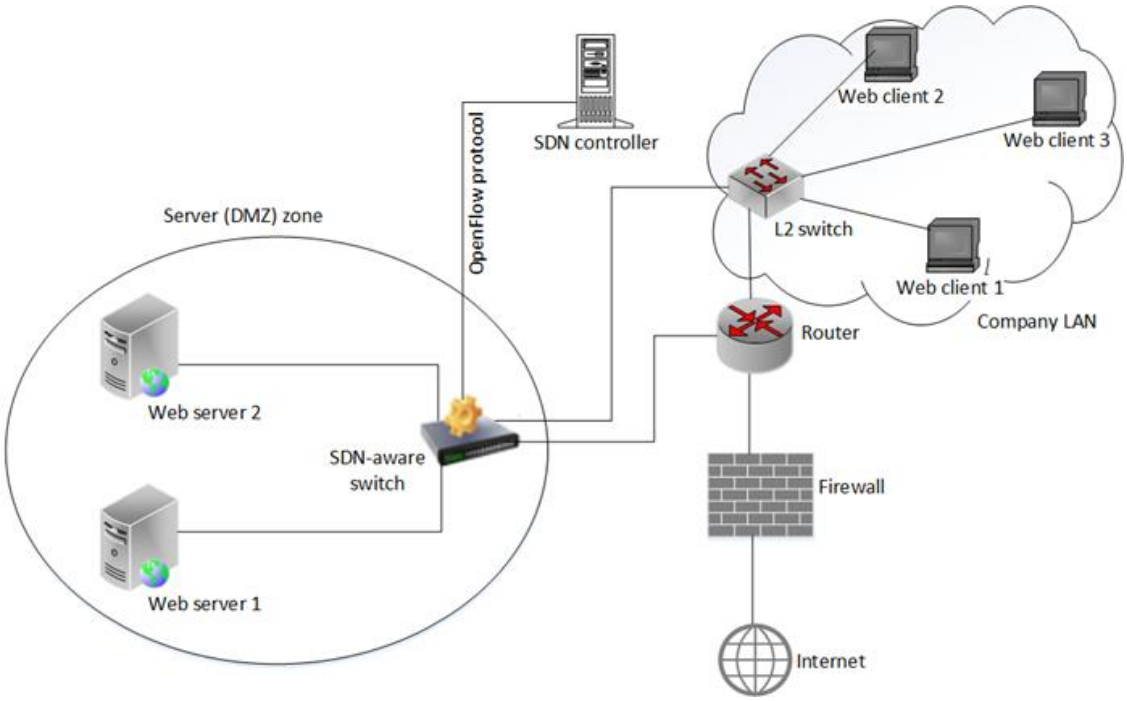
Mehanizam balansiranja opterećenja server mašina realizovan je na SDN kontroleru. Detaljnije objašnjenje koje se odnosi na tehnologiju balansiranja opterećenja biće izloženo u sledećem poglavlju.

Prikupljanje podataka o opterećenju ili zauzetosti resursa obavlja se uz pomoć SNMP protokola, kako bi se identifikovao server sa najmanjim opterećenjem.

SDN svič, po uputstvima SDN kontrolera obavlja zadatak balansiranja saobraćaja u SDN delu mreže. Sav ostali saobraćaj odvija se na isti način kao i kod tradicionalnih mreža. Predloženo rešenje je primenjivo na bilo kojim klijent-server okruženjima.

Zadatak SDN kontrolera jeste da formira instrukcije toka nakon obrade svakog prvog paketa jedne veze. Prateći ove instrukcije, svič bi trebalo da samostalno prosleđuje saobraćaj (sve pakete) jedne veze.

Rešenje sa slike 4.1 teži da omogući poboljšanu distribuciju korisničkih upita tako što primenjuje dinamičko upravljanje mrežom i time čini mrežu sposobnijom da usluži veći broj istovremenih korisnika. Pored toga, analiziranjem više parametara opterećenja resursa na serverima, ovaj pristup doprinosi da se umanji vreme odziva serverskih mašina u LAN mrežama i *data* centrima.



Slika 4.1 Primer hibridne SDN mreže sa minimalnim brojem SDN komponenti

5. Tehnologije balansiranja opterećenja

Opterećenje mreže je u praksi ublaženo kroz poboljšanje njene efikasnosti i pouzdanosti. Parametri koji se uglavnom optimizuju su proširivost, propusni opseg i vreme odziva. Pored toga i korišćenje resursa, ali i opterećenost bilo kakvih resursa redukuju se u cilju optimizacije rada mreže [51].

Tradicionalno balansiranje saobraćaja može biti statičko, dinamičko, ili kombinacija oba. Može biti realizovano na četiri načina [52]:

1. korisnici prikupljaju podatke o raspoloživosti servera (parametre) kako bi zakazali obradu svog upita i ostvarili ravnomernu opterećenost mreže;
2. postupak obrnutog dodavanja *proxy* servera, gde se brzina pristupa serverima, ali i balansiranje opterećenja poboljšavaju zahvaljujući keširanim podacima o prethodnim korisničkim vezama;
3. više server mašina različitih IP adresa dele isti DNS (eng. *Domain Name System*) server, pri čemu DNS server koristi interni mehanizam za ravnomernu raspodelu korisničkih upita nad server mašinama. Iako je ova metoda jednostavna, DNS server nema uvid u parametre opterećenosti individualnih server mašina (svi se ravnopravno posmatraju, bez obzira na njihovo stanje) [53];
4. saobraćaj se raspoređuje po server mašinama, na osnovu komunikacije između klijenata i servera za balansiranje opterećenja. Server za balansiranje opterećenja zadužen je za raspoređivanje upita ka server mašinama [54]. Ovaj pristup ume biti skup, jer često zahteva dodatnu fizičku arhitekturu.

5.1 Statičko i dinamičko balansiranje opterećenja

U praksi postoje dve kategorije mehanizama za balansiranje opterećenja zasnovane na trenutnom stanju sistema, a to su statičko i dinamičko balansiranje opterećenja [55].

5.1.1 Statičko balansiranje opterećenja

Pod statičkim balansiranjem opterećenja podrazumeva se mehanizam koji balansiranje vrši na osnovu predefinisane grupe pravila, tj. na ova pravila nikako ne utiče trenutno stanje sistema [56]. Glavna prednost ovih mehanizama jeste jednostavnost njihove implementacije. Sa druge strane, oni nemaju uvid u trenutno stanje fizičkih resursa mašina čije se opterećenje balansira, dakle, nisu svesni parametara poput zauzetosti kanala komunikacije, procesorskog vremena, zauzetosti kratkotrajne i dugotrajne memorije, itd. Ovi mehanizmi, ili kako se još nazivaju u literaturi algoritmi, nisu u stanju da podrže sisteme gde je potrebno prilagoditi mehanizam balansiranja trenutnom stanju sistema. To znači da su statičke metode balansiranja opterećenja pogodne u sistemima gde ne dolazi do značajnijih varijacija opterećenja (ne dolazi do značajnijeg porasta ili opadanja broja korisnika, ili do proširenja sistema novim hardverskim resursima i slično). Za statičke mehanizme takođe važi da jednom kada se „opterećenje“ dodeli konkretnoj mašini, više se ne može raspodeliti na druge [57].

Postoje optimalne i suboptimalne metode statičkog balansiranja [58]. Optimalne metode su zadužene za obezbeđivanje najmanje utrošenog vremena za donošenje odluke po pitanju balansiranja opterećenja. Kod suboptimalnih metoda to nije slučaj, one jednostavno donose odluku koja ne mora biti optimalna. Suboptimalne metode se primenjuju kod mehanizama koji nisu u stanju da pronađu optimalnu odluku u prihvatljivom intervalu vremena. U ovakvim slučajevima mogu se primenjivati i funkcije koje pomažu sistemu da donosi razumne odluke prilikom svake sledeće dodele resursa.

Prema arhitekturi, statički mehanizmi balansiranja opterećenja mogu se podeliti na centralizovane i distribuirane [59]. Dok se kod centralizovanog pristupa algoritam implementira na jednom čvoru (mašini u sistemu), distribuirani pristup podrazumeva rasprostranjenost algoritma za balansiranje na više čvorova. Centralizovan pristup je jednostavnije tehnički realizovati, ali mu je mana to što se čitav mehanizam oslanja na dostupnost samo jedne mašine u sistemu, dok kod distribuiranog pristupa taj problem ne postoji. Pogodniji pristup za realne sisteme na *cloud*-u je sigurno distribuirani.

Statički mehanizmi balansiranja pogodni su za relativno male sisteme, gde ima manje resursa koji nisu podložni promenama. Dakle, nisu predviđeni da se koriste u distribuiranim računarskim sistemima, jer ne umeju da izračunaju raspoloživost resursa, čak nemaju uvid ni u raspoloživost samih

server mašina čije se opterećenje balansira. U nastavku slede primeri postojećih mehanizama za statičko balansiranje.

Min-Min

Min-Min algoritam [60] kreira raspored sa minimalnim trajanjem procesa opsluživanja korisnika (u poređenju sa drugim tradicionalnim pristupima iz literature). Ipak, nedostatak ovog mehanizma jeste što sam raspored nije optimizovan po pitanju balansiranja opterećenja resursa. Drugim rečima, ovim algoritmom se ne obraća pažnja na ravnomernu opterećenost resursa, mada u literaturi postoje pokušaji prevazilaženja ovog nedostatka [60].

Max-Min

Max-Min algoritam [61] kreira takav raspored da se dužim zadacima daje veći prioritet. Na taj način je moguće izvršiti nekoliko kraćih zadataka u paraleli dok se izvršava neki duži zadatak. Pri izvršavanju ovog algoritma može se desiti da se upravo najduži zadatak izvršava korišćenjem „sporijih“ resursa, dok se „brži“ resursi koriste za paralelno izvršavanje kraćih i jednostavnijih zadataka. Dakle, i ovde postoji problem neravnomernog opterećenja resursa.

Oportunističko balansiranje opterećenja

Oportunističko balansiranje opterećenja podrazumeva proizvoljno raspoređivanje zadataka serverima. Prvi server koji je u mogućnosti da se odazove na korisnički zahtev to i čini [62]. Nije predviđen za realistična *cloud* okruženja, jer proces opsluživanja korisnika traje predugo.

Najkraće vreme izvršavanja

Ovaj mehanizam koristi parametar najkraćeg vremena izvršavanja, koji je u statičkoj varijanti ovog algoritma unapred izračunat (predodređen) [62]. Osnovni nedostatak mehanizma jeste to što ne uzima u obzir spremnost server mašine da preuzme zadatak.

Najkraće vreme završetka

Najkraće vreme završetka dopunjava prethodni mehanizam na taj način što uzima u obzir prosečno vreme spremnosti mašine za izvršavanje novog zadatka [62]. Zadatak se dodeljuje jezgru mašine sa najkraćim vremenom završetka zadatka.

Round Robin

Round Robin je vrlo jednostavan mehanizam koji teži da ravnomerno raspodeli opterećenje servera i takođe je najčešći statički mehanizam balansiranja u literaturi [63]. Najpre se formira lista svih dostupnih servera, a potom se svim serverima redom dodeljuju korisnički zahtevi. Na kraju liste se naredni korisnički zahtev ponovo dodeljuje prvom serveru u listi.

Postoji naprednija verzija ovog mehanizma, a to je *Weighted Round Robin* (WRR) mehanizam [64]. Razlika u odnosu na klasičan RR mehanizam jeste ta da se serverima dodeljuju težinski faktori, na osnovu kojih se raspoznaje kapacitet svakog od servera. Kada konkretan server poseduje višu vrednost težinskog faktora, to znači da može da prihvati više istovremenih korisničkih zahteva od servera sa nižim vrednostima težinskog faktora. Ako na primer u datom sistemu sa dva servera, jedan server ima težinski faktor jednak 4, a drugi težinski faktor jednak 1, onda će prvom serveru biti dodeljena prva četiri korisnička zahteva, pa će tek peti zahtev biti dodeljen drugom serveru. Iz opisa metode može se zaključiti da je upotrebljivija od klasičnog RR samo u slučaju servera sa nejednakim kapacitetima.

Iako je ovaj algoritam jednostavan za implementiranje, u slučajevima velikih sistema i kada je opterećenje servera korisničkim zahtevima suviše promenljivo, ne dolazi do efikasnog balansiranja.

Prilikom testiranja rešenja za balansiranje opterećenja koje ovo istraživanje nudi, za međusobno poređenje performansi korišćen je između ostalih i RR mehanizam, kao najzastupljeniji statički algoritam u literaturi, ali i praksi.

5.1.2 Dinamičko balansiranje opterećenja

Za razliku od statičkog balansiranja opterećenja, dinamičko balansiranje saobraćaja ne podleže unapred definisanim pravilima, već se ista kreiraju u skladu sa različitima zahtevima [55]. Ovakvi mehanizmi balansiranja omogućavaju da se preusmeravanjem saobraćaja koriguje neuravnoteženost, u slučaju pojave neravnomernog opterećenja na server mašinama. Odgovarajući algoritmi ovih mehanizama nadgledaju i periodično prikupljaju vrednosti parametara opterećenja servera. Zatim, koriste prikupljene podatke kako bi proverili opterećenja server mašina i preduzeli odgovarajuće akcije preusmeravanja saobraćaja na druge server mašine u slučaju otkrivanja neuravnoteženog opterećenja. Sve ovo dovodi do poboljšanja performansi celog sistema.

Kod ove metode balansiranja saobraćaja, putanje po kojima se raspoređuje saobraćaj se dinamički menjaju (npr. u skladu sa aktuelnim potrebama sistema [56]). Zahvaljujući ovoj karakteristici, rešenje koje podrazumeva dinamičko balansiranje saobraćaja daleko je efikasnije u realnim sistemima i na *cloud*-u, jer se moderne mreže stalno proširuju novim korisnicima i servisima. Algoritmi dinamičkog balansiranja na efikasan način podižu performanse sistema, bez zastoja u radu mreže i nepotrebnih troškova.

Mehanizmi dinamičkog balansiranja opterećenja takođe se mogu podeliti na distribuirane i centralizovane. Prvi tip opisuje činjenica da svi čvorovi u mreži poseduju informacije o opterećenosti mašina i učestvuju donošenju odluka pri balansiranju opterećenja. Na ovaj način se ostvaruje otpornost sistema na greške nastale u jednoj tački sistema (na jednom čvoru). Čvorovi međusobno

sarađuju kako bi optimizovali proces balansiranja i to onda čini algoritme balansiranja kooperativnim (u suprotnom, algoritmi se kategorišu kao nekooperativni).

Centralizovani mehanizmi balansiranja podrazumevaju da se nadzor resursa i donošenje odluke vrše na jednom centralnom čvoru. Ovakva rešenja su sa jedne strane jednostavnija za implementaciju, a sa druge strane nose određeni rizik koji se vezuje za potencijalnu pojavu otkaza na tom centralnom čvoru. Međutim, uprkos riziku koji nosi sa sobom primena centralizovanog rešenja, nov mehanizam za balansiranje opterećenja, opisan ovim istraživanjem, zasniva se upravo na centralizovanom pristupu, jer se radi o sistemu sa minimalnim brojem komponenti (čvorova). Predloženo rešenje pogodno je za ispitivanje i u sastavu distribuiranog sistema, bez obzira na to što je ovim istraživanjem ispitano samo nad centralizovanim sistemom. Buduća istraživanja pokazaće otpornost ovog rešenja na greške i kod distribuiranih sistema.

Postoje i prelazne varijante rešenja, a to su poludistribuirani mehanizmi, gde se čvorovi grupišu u klastere, pri čemu je za balansiranje opterećenja u jednom klasteru odgovoran jedan (centralni) čvor tog klastera.

U svim modernim mehanizmima dinamičkog upravljanja balansiranjem opterećenja prepoznaju se naredne faze [58]:

1. nadzor opterećenja – posmatraju se dostupnost i opterećenje serverskih resursa,
2. sinhronizacija – razmena informacije o dostupnosti i opterećenju resursa,
3. balansiranje na osnovu kriterijuma – proračunava se kojem će serveru pripasti zadatak i
4. migracija zadatka – preusmeravanje zadatka na prethodno odabran server.

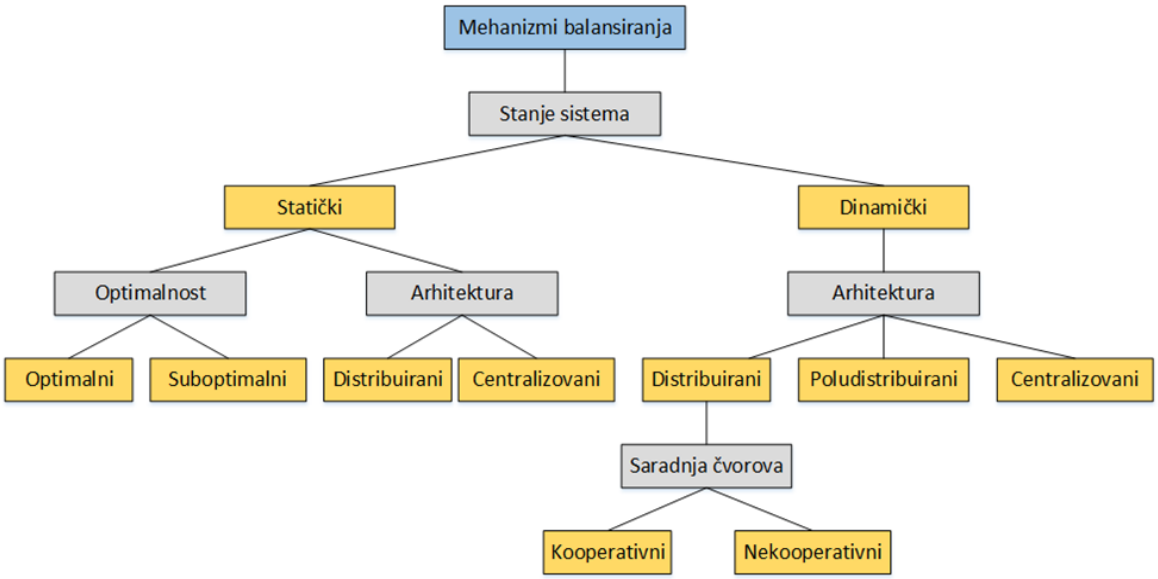
Ove faze ili koraci u mehanizmima dinamičkog balansiranja takođe prepoznaju nekoliko polisa ponašanja:

- Polisa prenosa – odgovara na pitanje pod kojim uslovima bi trebalo da zadaci budu premešteni sa jednog čvora na drugi. Nadolazeći zadaci bivaju preusmereni na drugi čvor ili obrađeni lokalno. Ova polisa zavisi od opterećenja na čvorovima i odgovara fazama balansiranja na osnovu kriterijuma i migracije zadatka.
- Polisa selekcije – razmatra koji će zadaci biti preusmereni. Uzima u obzir uslove pod kojima bi se preusmeravanje desilo, na primer količinu utroška vremena za migraciju i vreme potrebno za izvršenje zadatka.
- Polisa lokacije – proverava koji su čvorovi rasterećeni i preusmerava zadatke na njih. Takođe proverava da li su servisi spremni za prenos i koliko je izvodljivo obraditi zadatak na ciljanom čvoru.

- Polisa informacije – prikuplja sve informacije koje se tiču čvorova u sistemu i ostale polise je koriste za donošenje sopstvenih odluka. Trenutak prikupljana informacija se takođe definiše ovom polisom.

Uzimajući u obzir sve iznete činjenice, moguće je zaključiti da su dinamička rešenja balansiranja opterećenja u današnjim *cloud* sistemima zaduženim za opsluživanje više stotina miliona korisnika primenjiva, a statička nisu.

Na slici 5.1 izložene su sve klasifikacije i statičkog i dinamičkog balansiranja opterećenja iznete u ovom poglavlju.



Slika 5.1 Klasifikacija mehanizama za balansiranje opterećenja

Least Connection

LC (eng. *Least Connection*) [65] mehanizam je jedan od starijih pristupa dinamičkog balansiranja opterećenja servera. Ovim mehanizmom se meri trenutan broj uspostavljenih veza ka server mašinama usled korisničkih zahteva, a zatim se za opsluživanje datog korisnika bira server sa najmanjim brojem ostvarenih veza. Na centralnom čvoru se prati broj aktivnih veza svakog servera. U slučaju uspostavljanja nove veze broj aktivnih veza se povećava za jedan, a u slučaju kraja započete veze, taj broj se smanjuje za jedan.

Kao i za RR mehanizam i za LC mehanizam postoji varijanta rešenja sa težinskim faktorima – *Weighted Least Connection* (WLC). Težinski faktori mogu biti dodeljeni na osnovu različitih kriterijuma performansi servera. Serveri s većim težinskim faktorom će prihvatiti veći procenat veza u bilo kom trenutku u poređenju sa serverima sa manjim težinskim faktorom, a koji poseduju isti broj aktivnih veza. Podrazumevana vrednost faktora je 1, a administrator sistema ili program za nadzor mogu menjati vrednost faktora. Opsluživanje korisnika će pripasti serveru sa najmanjim odnosom broja aktivnih veza ka korisnicima i težinskog faktora.

U skorijem radu [65] predložena je nešto naprednija verzija LC mehanizma. Modifikacija se zasniva na algoritmu preopterećenja koji sprečava da se na novu server mašinu preusmeri ogroman broj korisničkih zahteva.

Neprestano praćenje paketa koji označavaju početak i kraj jedne veze može prouzrokovati zagušenje saobraćaja na centralnom čvoru, jer se zapravo svaki paket veze mora slati tom čvoru kako bi se parsirao i proverio da li je on prvi ili poslednji paket veze.

5.2 Balansiranje opterećenja u SDN mrežama

SDN kontroler je uređaj sa ključnom ulogom u SDN mreži. Zadužen je za prosleđivanje svakog novog toka podataka i odgovoran je za odabir servera sa najmanjim opterećenjem u realnom vremenu [66]. Međutim, posedovanje centralnog uređaja i smeštanje celokupne logike na njega, stvara probleme sa pouzdanošću, proširivosti i odzivom sistema [67].

Statičkim pristupom u balansiranju saobraćaja uvek se dobija isti ishod za specifičan unos ulaznih parametara. Vrednosti parametara i početna situacija u sistemu daju ishod. Dinamički pristup se zasniva na algoritmu, na osnovu kojeg je moguće dobiti različite ishode za iste vrednosti ulaznih parametara. Glavne prednosti i mane oba pristupa izložene su u tabeli 5.1.

	Statički pristup	Dinamički pristup
Prednosti	• bolji propusni opseg • bolji stepen ravnomernog opterećenja • bolje vreme odziva • bolji troškovi	• bolji stepen ravnomernog opterećenja • bolja iskorišćenost resursa • bolje kašnjenje
Mane	• neprihvatljiva potrošnja energija • neprihvatljivo kašnjenje • neprihvatljiva dostupnost • neprihvatljiv gubitak paketa	• neprihvatljiv propusni opseg • neprihvatljiva potrošnja energije • neprihvatljiv gubitak paketa • neprihvatljivi troškovi

Tabela 5.1 Glavne prednosti i mane kod statičkog i dinamičkog pristupa [51]

Nezavisno od samog pristupa implementiranog u mreži, moraju se definisati odgovarajuće metrike. Razni kvalitativni parametri mogu se upotrebiti za procenu ravnomernog opterećenja u SDN mreži:

- iskorišćenost resursa (optimalan algoritam za maksimizaciju iskorišćenosti resursa [68]),

- kašnjenje (algoritmi za direktno rutiranje minimizuju kašnjenje [69]),
- učestanost gubitaka paketa,
- vreme odziva,
- propusnost,
- troškovi migracije,
- opterećenje u radu (balansiranje opterećenja u radu među kontrolerima [70]),
- potrošnja energije (odabir balansiranja opterećenja tako da se redukuje potrošnja energije [71]),
- prosleđivanje upita (neophodno je smanjiti broj upita kako bi se sačuvali memorijski resursi [72]) i
- vreme izvršavanja.

U [73] su sprovedena istraživanja u vezi sa višim stepenom programabilnosti i dinamičnosti mreža, gde se opisuje balansiranje opterećenja korišćenjem algoritama za podelu saobraćaja klijenata na osnovu važnosti. Sa druge strane u [74] opisuje se mogućnost višestrukog balansiranja opterećenja za različite servise (na primer jedno balansiranje za saobraćaj koji potiče od pretraživanja interneta i drugo za e-mail servis). Zatim, u radu [75] naglašava se problem znatnog pogoršanja rada servisa usled neravnomernog opterećenja u mreži nastalog kao posledica upotrebe više SDN kontrolera. Kao rešenje nudi se implementacija balansiranja opterećenja zasnovana na tehnici celobrojnog linearnog programiranja (eng. *Instruction-level parallelism*, ILP). Konačno rad [76] predlaže balansiranje opterećenja na osnovu novog algoritma koji sadrži koncept kažnjavanja baziranog na opterećenju kako bi optimizovao dodelu odgovornosti kontroleru (od strane sviča) i postigao kompromis između povratne putanje (eng. *Round-trip Time* , RTT) i opterećenja kontrolera.

5.2.1 Balansiranje opterećenja u hibridnim SDN mrežama

Prethodno je naglašeno da implementacija SDN rešenja nije jednostavna, te hibridna SDN mreža predstavlja prirodno privremeno rešenje. Osnovni cilj istraživanja, koje je sprovedeno u okviru ove disertacije, jeste definisanje efikasnog metoda primene hibridne SDN mreže i to kroz nuđenje konkretnog modela čiji će rezultati biti prezentovani niže u tekstu. U praksi već postoje određena rešenja. Neka od njih se kao u [77] zasnivaju na primeni inovativnih metoda rutiranja i raspoređivanju tokova podataka na osnovu balansiranih opterećenosti procesora servera i veze ka serveru. Primena ovakvog rešenja „opterećena" je određenim ograničenjima, koja nastaju prvenstveno zato što se pažnja uglavnom posvećuje problemu zagušenja saobraćaja u mreži usled pojave *Open Shortest Path First* (OSPF). U [78] razmatran je mehanizam razdvajanja pri upravljanju tokovima, gde putanja

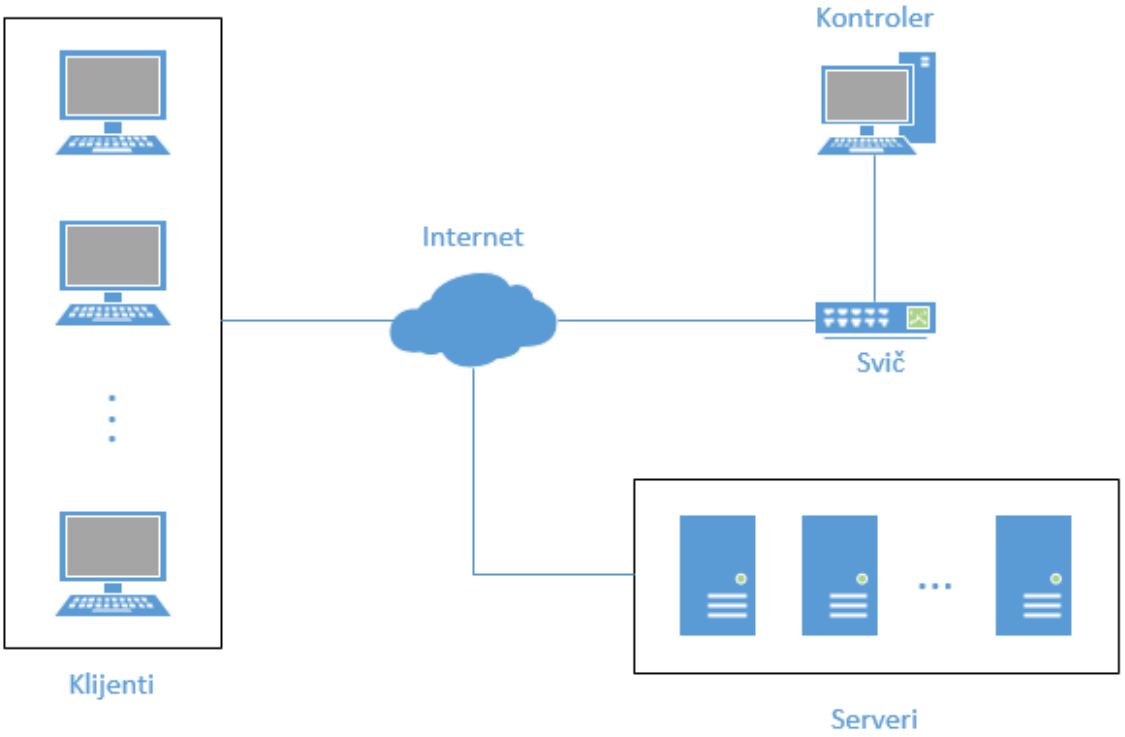
svakog toka podataka mora prolaziti kroz SDN svič kako bi se postigla bolja kontrola tokova i upravljanje saobraćajem. Kao metrike koriste se iskorišćenost veze i kašnjenje. Pošto se kod oba predložena rešenja ne uzima u obzir kvalitet servisa, može doći do neefikasne obrade rutiranja.

U prethodnim pristupima nije razmatrana primena svih parametara kvaliteta servisa za optimizaciju opterećenja. Dakle, bilo bi od interesa ispitati kako odabrati odgovarajuće parametre za kvalitet servisa tako da se optimizuju odluke balansiranja saobraćaja u različitim scenarijima. Neka od istraživanja ispituju mogućnosti razvoja hibridne SDN mreže u okviru *data* centara. U radu [79] realizovano je efikasno prosleđivanje saobraćaja kroz rutiranje koje uzima u obzir kvalitet servisa time što su metrike zasnovane na gubicima paketa, kašnjenju, i propusnosti. Osnovna mana ovog pristupa jeste visoka kompleksnost neophodnih računskih operacija i kašnjenje usled povećanja potrebne procesorske moći.

Ozbiljno ograničenje mnogih ponuđenih rešenja za balansiranje opterećenja jeste izostanak kontinualne detekcije opterećenja resursa u procesu odlučivanja kojem serveru će pripasti dužnost opsluživanja korisnika. Kontinualnu detekciju opterećenja moguće je izvesti periodičnim testiranjem parametara QoS-a. Međutim, svako takvo testiranje utiče na sam QoS, te merni mehanizam može negativno uticati na efikasnost balansiranja. Na primer, propusni opseg se ispituje tako što se zauzme ceo opseg, što za posledicu ima remećenje redovnog saobraćaja. Važnost ovakvog ograničenja ogleda se u prirodi novih komunikacionih tehnologija (poput IoT i M2M) i neprekidnom rastu broja internet korisnika. Iako rešenja predlažu balansiranje saobraćaja korišćenjem metrika poput kvaliteta veze, propusnog opsega i RTT, nedostaje im osvrt na resursne zahteve servisa. Još jedan od ciljeva ovog istraživanja jeste razvoj rešenja za balansiranje opterećenja koje bi bilo prilagodljivo resursnim potrebama različitih servisa. Drugim rečima, ovo rešenje bi bilo moguće konfigurisati na osnovu iskorišćenosti resursa za dati servis.

Load Balancing by Server Response Time

LBBSRT je skoriji mehanizam za dinamičko balansiranje opterećenja [6], dodeljuje korisničke zahteve serverima na osnovu vremena odziva servera. S obzirom da je ovo relativno nova metoda i da je pokazala neke dobre rezultate, ona je odabrana za poređenje sa LBORU metodom i to posebno u delu istraživanja u kojem se analiziraju performanse predloženog LBORU mehanizma za balansiranje opterećenja. Na slici 5.2 prikazan je model sistema koji je korišćen pri testiranju LBBSRT algoritma u [6]. Iz čega se jasno može uočiti da je sličan sistemu korišćenom pri testiranju LBORU metode.



Slika 5.2 Model sistema pri testiranju LBBSRT metode u [6]

Procedura dobavljanja informacije o vremenu odziva servera je sledeća:

1. Na početku rada sistema kontroler šalje *OpenFlow*–ov *PacketOut* paket sviču koji sadrži vremenski interval t. Kontroler pamti vreme slanja paketa. Broj *PacketOut* poruka koje se šalju jednak je broju raspoloživih server mašina. Kao IP adresa izvora *PacketOut* poruke navedena je adresa kontrolera, dok je kao IP adresa destinacije navedena adresa datog servera.
2. Po prijemu *PacketOut* paketa, svič parsira ove pakete i šalje pakete sa parsiranim podacima svim odgovarajućim serverima.
3. Server šalje poruku odgovora iz koje kontroler kasnije izvlači podatak o vremenu odziva servera. Da bi ova poruka stigla do kontrolera, server simulira pravi korisnički zahtev tako što šalje paket sa podacima, pri čemu je adresa izvora paketa postavljena na IP adresu servera, dok je adresa destinacije postavljena na IP adresu kontrolera. Pošto je reč o novom događaju u tabeli tokova, svič mora da pošalje *PacketIn* poruku kontroleru. Na ovaj način kontroler konačno dobija podatak o vremenu pristizanja paketa sa servera tako što parsira *PacketIn* poruku. Kontroler je takođe u tom momentu u stanju da izračuna vreme odziva servera i upiše taj podatak u lokalnu bazu.
4. Koraci 1, 2, 3 mogu se ponavljati dokle god se na početku rada sistema ne završi podizanje sistema.

Sledi opis algoritma dobavljanja vremena odziva servera [6]. Jasno je da merenje vremena odziva daje i mnogo intuitivniju ocenu o serveru, a to je informacija o tome da li je server uopšte dostupan.

Algorithm1: Measure server's response time

1. **While** system startup **do**
2. **If** current time % t == 0 **do**
3. Send Packet_out to switches and record sending time T_{send} ;
4. **End if**
5. **If** receive a Packet_in message **then**
6. Parse message;
7. **If** the source address of the received packet is the server, the destination address is the controller **then**
8. Record the time T_{arrive} of received message;
9. Calculate the response time by the formula $T_{response} = T_{arrive} - T_{send}$;
10. Store response time;
11. **Else**
12. Send to other modules;
13. **End if**
14. **End if**
15. **End while**

Algoritam 5.1 Merenje vremena odziva servera [6]

Postupak balansiranja opterećenja uz pomoć vremena odziva servera definisan je sledećim koracima:

1. Korisnici na početku šalju ARP (eng. *Address Resolution Protocol*) pakete svim svičevima sa kojima su povezani, pa tako ti paketi dospevaju i na svič povezan sa kontrolerom (svič na slici 5.2). ARP paketi se šalju jer je to prvi pristup korisnika serverima, te svič nema zabeležene potrebne podatke u svojoj tabeli tokova i ne može drugačije da pošalje *PacketIn* poruku kontroleru. Nakon toga kontroler kreira virtuelnu MAC adresu na osnovu koje šalje *PacketOut* poruku sviču, a svič šalje ARP poruku odgovora korisniku.
2. Po prihvatanju ARP poruke, korisnik šalje zahtev za servis serveru. Kontroler će primiti ovaj paket na sličan način kao u 1. koraku i za opsluživanje korisnika odabrati server sa najkraćim ili stabilnim vremenom odziva na osnovu prethodno prikupljenog podatka od strane servera. U nastavku sledi opis algoritma za odabir servera koji će opslužiti datog korisnika:

a) Izrazima (1) i (2) određuju se maksimalno i minimalno vreme odziva servera za datu grupu raspoloživih servera.

$$T_{max} = \max\{T_{1,0}, T_{2,0}, \dots, T_{n,0}\} \quad (1)$$

$$T_{min} = \min\{T_{1,0}, T_{2,0}, \dots, T_{n,0}\} \quad (2)$$

b) Na osnovu T_{min} i T_{max}, izračunava se $| T_{min} - T_{max} |$. Sa λ se označava raspon vrednosti vremena odziva, kada su serveri sličnog opterećenja. Ukoliko je $| T_{min} - T_{max} | > \lambda$, bira se server sa vremenom odziva T_{min}, u suprotnom se računa standardna devijacija vremena odziva svakog od servera, tako što se izračunava standardna devijacija prethodnih m vremena odziva korišćenjem izraza (3).

$$S_i = \sqrt{\left(T_{i,0} - \overline{T}\right)^2 + \left(T_{i,1} - \overline{T}\right)^2 + \dots + \left(T_{i,m-1} - \overline{T}\right)^2} \quad (3)$$

gde, $\overline{T}$ predstavlja srednju vrednost prethodnih m vremena odziva, S_i je standardna devijacija prethodnih m vremena odziva i-tog servera. Potom se odabere server sa najmanjom vrednošću standardne devijacije S_{min}.

c) Na kraju kontroler šalje sviču odgovarajući zapis za tabelu tokova na osnovu odabranog servera, te se korisnički zahtev može proslediti odabranom serveru.

Pseudo kod za dodeljivanje korisničkog zahteva određenom serveru dat je algoritmom 5.2 [6].

Algorithm2: Handle user requests

1. **If** the controller receives a Packet_in message **then**
2. Parse message;
3. **If** data package for ARP request **then**
4. Controller sends Packet_out message reply ARP;
5. **End if**
6. **If** packet for user service request **then**
7. Based the formula (1), (2) obtain the current server in the cluster server response time maximum and minimum value, T_{max} , T_{min} ;
8. **If** $|T_{min} - T_{max}| < \lambda$ **then**
9. By the formula (3) select the standard deviation of the minimum value corresponding to the server;
10. **Else**
11. Select the T_{min} corresponding to the server;
12. **End if**
13. **Else**
14. Send to other modules;
15. **End if**
16. **End if**

Algoritam 5.2 Rukovanje zahtevima korisnika [6]

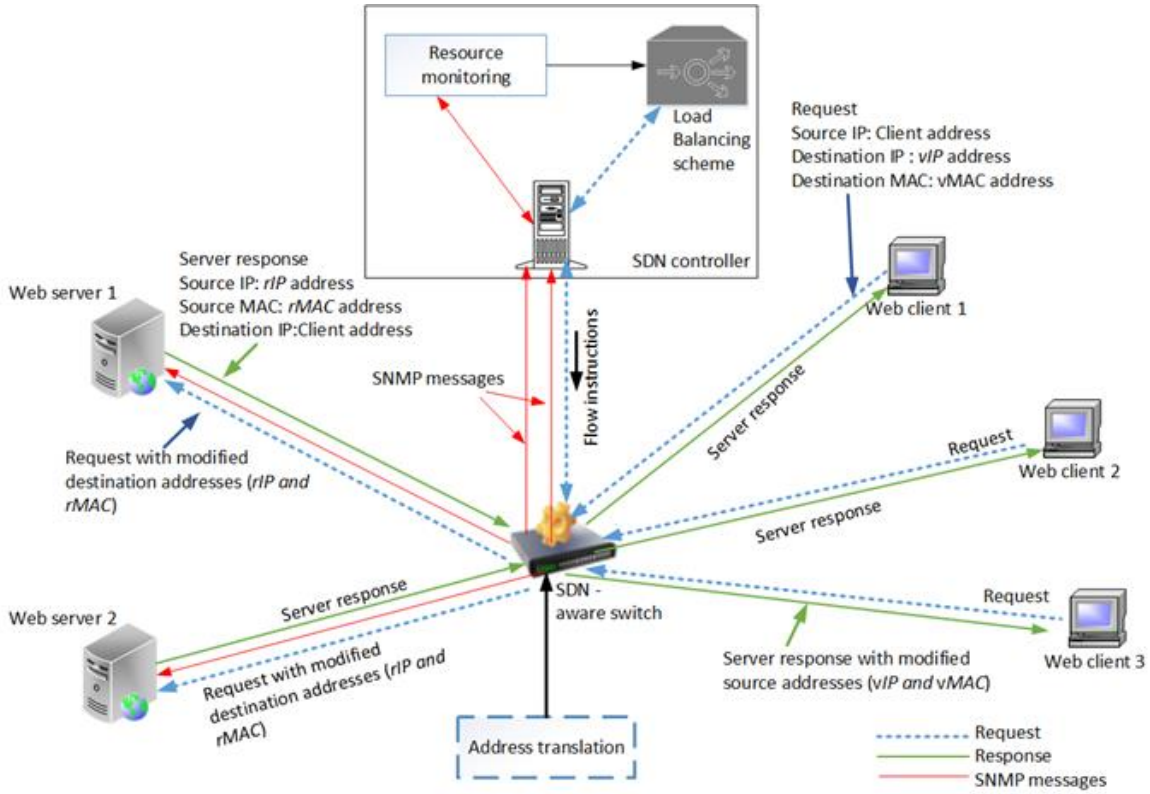
Ograničenje LBBSRT mehanizma jeste energetska neefikasnost. Autori rada [6] navode da bi trebalo optimizovati potrošnju energije kad je u pitanju celokupan algoritam za balansiranje opterećenja servera.

5.3 Nova šema balansiranja opterećenja - LBORU metoda

Predlog rešenja izloženog u ovom istraživanju sadrži protokol komunikacije između klijenata i servera koji konfigurišu u SDN delu hibridne SDN mreže. Osnovna ideja rešenja jeste da svi serveri čije se opterećenje balansira dele jednu virtuelnu IP adresu (vIP). Ta IP adresa dodeljena je serverima od strane SDN kontrolera. Pored virtuelne IP adrese potrebno je da serveri dele i jednu virtuelnu MAC adresu (vMAC). Mapiranje vIP adrese na vMAC adresu je nepromenljivo, kako bi se izbegao problem sa ARP keširanjem koji može degradirati performanse balansiranja opterećenja.

vIP adresa je destinaciona adresa paketa koji potiču od klijenata. Klijenti šalju zahteve na ovu adresu i tako zahtevaju konkretne resurse od servera. Kako klijent pošalje zahtev, tako SDN svič presretne isti taj zahtev i preusmeri ga SDN kontroleru. SDN kontroler je taj koji je zadužen za

prosleđivanje klijentskog zahteva određenom serveru. Kada su u pitanju već postojeće veze između klijenata i servera SDN svič jednostavno prosleđuje sve pakete na osnovu zapisa u tabeli tokova koja se nalazi na SDN sviču. Na ovaj način je rešenje za balansiranje opterećenja implementirano na SDN kontroleru. Šema prethodno opisanog protokola predstavljena je slikom 5.3.



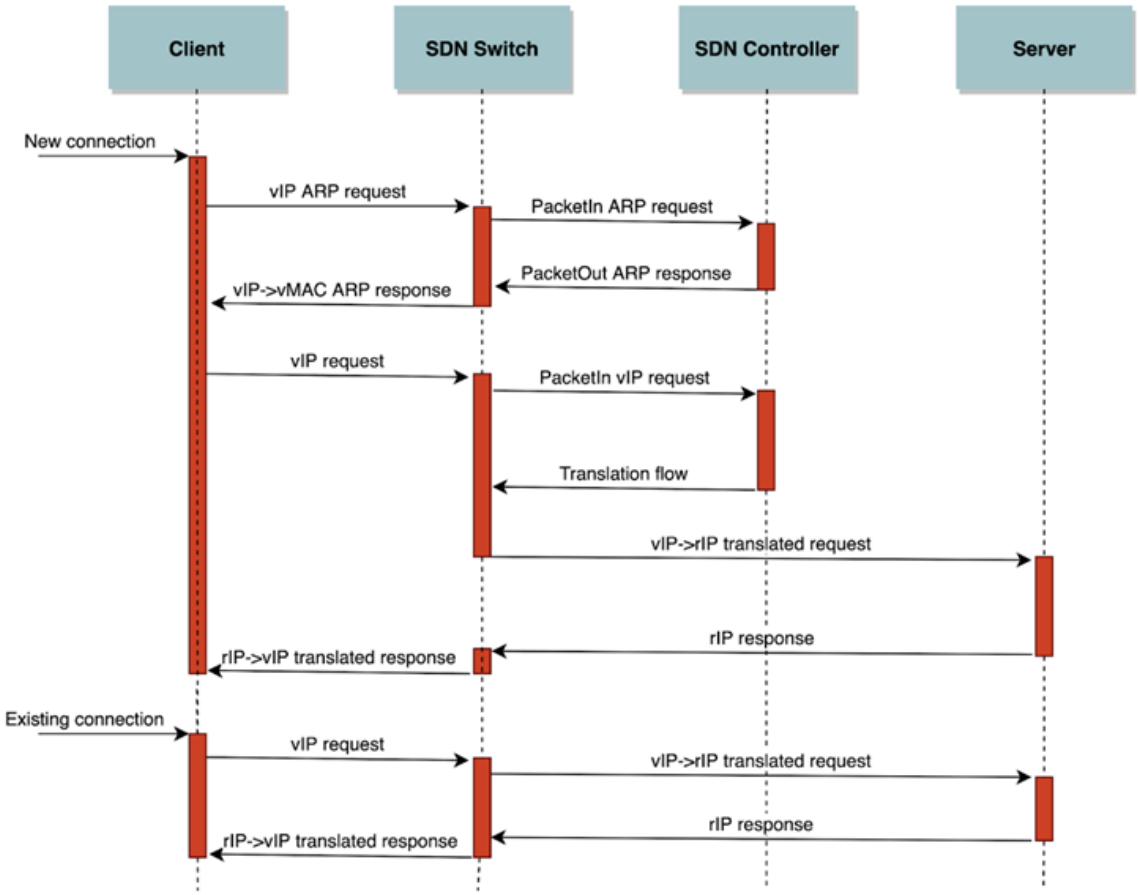
Slika 5.3 Šema predloga za balansiranje opterećenja servera

SDN kontroler odlučuje o preusmeravanju zahteva na određen server tako što prvo prihvati prvi paket uspostavljene veze sa korisnikom (*OpenFlow*, *PacketIn*), potom detektuje server sa najmanjim opterećenjem i na kraju kreira odgovarajuće instrukcije toka podataka (*OpenFlow PacketOut*). Na osnovu ovih instrukcija SDN svič postaje zadužen za:

a) modifikovanje paketa, odnosno izmenu destinacionih vIP i vMAC adresa u realne adrese (rIP i rMAC) server mašine sa najmanjim opterećenjem,
b) prosleđivanje modifikovanog paketa istom tom serveru,
c) po prijemu odgovora od odabranog servera, ponovno prevođenje rIP i rMAC u vIP i vMAC adrese paketa koji predstavlja odgovor servera i
d) prosleđivanje modifikovanog odgovora servera (paketa) klijentu.

SDN svič kešira (čuva) podatke izvršenih prevođenja na određen period. Na sviču se obrađuje svaka već postojeća veza sa klijentom zahvaljujući tim keširanim podacima. Ovi podaci su raspoređeni u tabele tokova na SDN sviču. Na SDN kontroleru postoje dva neprekidna procesa: periodično prikupljanje informacija o zauzetosti (opterećenju) resursa server mašina preko SNMP

protokola i odlučivanje o usmeravanju saobraćaja ka konkretnom serveru. Dijagram toka razmena poruka predstavljen je na slici 5.4.



Slika 5.4 Dijagram toka razmene poruka

Algoritam za prikupljanje informacija, odnosno parametara o opterećenju pojedinačnih servera dat je algoritmom 5.3. Jedna iteracija algoritma odgovara prikupljanju parametara opterećenja jednog servera. SDN kontroler najpre proveri dostupnost SNMP protokola na datom serveru. Ukoliko je server nedostupan, jednostavno se prelazi na sledeću iteraciju, odnosno server. Ukoliko je server dostupan, SDN kontroler prikuplja informacije o opterećenosti procesora (*CPU Load*), broju pristupa disku pri čitanju sa diska (*I/O Read*), broju pristupa disku pri pisanju na disk (*I/O Write*), količini poslatih podataka (*Link Upload*) i količini primljenih podataka (*Link Download*). Ovi podaci se prikupljaju SNMP protokolom, a skladište se u lokalnu promenljivu nalik matrici. Potom algoritam prelazi na rad sa sledećim serverom i tako se postupak ponavlja sve do poslednjeg servera na raspolaganju, gde mu je i kraj.

Algorithm 3: SNMP Algorithm

Input: servers - set of servers $(S_1, S_2, \ldots, S_N)$
Output: metrics - set of metrics for each available server

```
for server = S_1, S_2, ..., S_N do
    if SNMP_is_available(server) then
        cpu = SNMP_get(server, CPU_OID)
        ioread = SNMP_get(server, IOREAD_OID)
        iowrite = SNMP_get(server, IOWRITE_OID)
        download = SNMP_get(server, LINK_DOWNLOAD_OID)
        upload = SNMP_get(server, LINK_UPLOAD_OID)

        metrics(server) = (cpu, ioread, iowrite, download, upload)
    end if
end for
```

Algoritam 5.3 SNMP algoritam

Algoritam se periodično ponavlja kako bi se ažurirali podaci o opterećenosti. Važno je istaći da treba pažljivo odabrati taj interval vremena. Sa jedne strane, interval mora biti dovoljno kratak da bi se podaci na vreme ažurirali i bili dovoljno pouzdani. Sa druge strane, ukoliko je taj interval suviše kratak, može da se desi da dođe do zagušenja u saobraćaju (usled opterećenja na vezi između SDN kontrolera i server mašina). Kao kompromis, u radu je odlučeno da se koristi interval od jedne sekunde. Da bi se postiglo brže balansiranje opterećenja potrebno je koristiti vreme kraće od jedne sekunde.

Nakon prikupljanja podataka prethodno opisanim algoritmom, SDN kontroler mora da izvrši međusobno poređenje odgovarajućih parametara opterećenja svakog od servera. Za potrebe poređenja neophodno je formirati matricu *A* (4) od prikupljenih parametara opterećenja čiji su elementi a_{ij}, gde je *i* redni broj parametra (1 – *CPU Load*, 2 - *I/O Read*, 3 - *I/O Write*, 4 - *Link Upload*, 5 - *Link Upload*), a *j* redni broj servera (*j* = 1, 2, ..., *n*).

$$A = \left\| a_{ij} \right\|_{(5*n)} = \begin{vmatrix} a_{cpu\,1} & a_{cpu\,2} & a_{cpu\,3} & \cdots & a_{cpu\,n} \\ a_{ior\,1} & a_{ior\,2} & a_{ior\,3} & \cdots & a_{ior\,n} \\ a_{iow\,1} & a_{iow\,2} & a_{iow\,3} & \cdots & a_{iow\,n} \\ a_{upl\,1} & a_{upl\,2} & a_{upl\,3} & \cdots & a_{upl\,n} \\ a_{dwl\,1} & a_{dwl\,2} & a_{dwl\,3} & \cdots & a_{dwl\,n} \end{vmatrix} \quad (4)$$

Za svaki od parametara traži se server ili serveri (ukoliko ih ima više) sa najmanjom vrednošću tog parametra i potom se u zavisnosti od tih minimalnih vrednosti svakom elementu a_{ij} iz (4) pridružuje još jedna vrednost b_{ij} (5) na osnovu sledećeg pravila:

$$b_{ij} = \begin{cases} 1, & a_{ij} = \min\limits_{j=1,\dots,n}(a_{ij}) \\ 0, & a_{ij} > \min\limits_{j=1,\dots,n}(a_{ij}) \end{cases} \tag{5}$$

Na ovaj način moguće je formirati jednu binarnu matricu *B* (6) koja je istih dimenzija kao matrica *A* (4). Matrica *B* sadrži broj bodova dodeljen serverima za svaki od njihovih pet parametara.

$$B = \|b_{ij}\|_{(5*n)} \tag{6}$$

Svaki od redova matrice *B* može sadržati jedan ili više elemenata b_{ij} različitih od nule. Cilj je pronaći server sa najmanjim opterećenjem u datom trenutku. Dakle, neophodno je istaći da nemaju svi parametri podjednak uticaj na performanse svakog servisa. Uticaj svakog od parametara se stoga mora valorizovati. Drugim rečima, u zavisnosti od tipa servisa potrebno je definisati težinske koeficijente parametara k_i tako da je $k_i \in [0, 1]$ za svako $i = 1, 2,..., 5$. Podrazumevana vrednost k_i je 1 i može biti modifikovana kako bi se uticaj parametra prilagodio zahtevima servisa. Na primer, u slučaju balansiranja opterećenja baze podataka, koeficijenti k_2 (*I/O Read*) i k_3 (*I/O Write*) bi bili od veće važnosti od preostalih koeficijenata kako bi se postigle bolje performanse i odzivi baze podataka. U tom slučaju mogle bi se primeniti sledeće vrednosti koeficijenata k_i: *CPU Load* = 0.5, *I/O Read* = 1, *I/O Write*= 1, *Link Download* = 0.2, *Link Upload* = 0.2.

Šema za balansiranje opterećenja definisana je tako da je server sa najmanjim opterećenjem server sa najvećim brojem bodova. Dakle, potrebno je sabrati ukupan broj bodova svakog od servera prema relaciji (7):

$$S_i = \sum_{i=1}^{5} k_i b_{ij}, \quad j = 1, \dots, n \tag{7}$$

Na kraju preostaje da se pronađe server sa najvećim brojem ostvarenih bodova, a to se čini tako što se pronađe maksimalna vrednost prethodno dobijenog vektora *S = [S1, S2,..., Sn]*. Dakle, traži se indeks servera sa najmanjim opterećenjem *W* (8):

$$W = \left\{ j \mid S_j = \max_{m=1,\ldots,n} (S_m) \right\} \tag{8}$$

Prema tome, dobijanjem indeksa *W*, SDN kontroler postaje svestan servera sa najmanjim opterećenjem i stoga je spreman da odluči kojoj rIP adresi treba da prosledi saobraćaj. SDN kontroler generiše instrukcije neophodne SDN sviču da izvrši prevođenje vIP adrese u rIP adresu.

6. Koncept rešenja

Rešenje ovog istraživanja, pored SDN koncepta, sadrži i koncept koji primenjuje tehnologiju balansiranja opterećenja (eng. *load balancing*). Povezivanjem ova dva koncepta dobija se koncept rešenja ovog istraživačkog rada, koji podrazumeva upotrebe SDN sviča kao *engine*-a i SDN kontrolera kao upravljača.

Upotreba vIP i vMAC detaljno je adresirana u prethodnom poglavlju (5.3 Nova šema balansiranja opterećenja - LBORU metoda). U nastavku poglavlja najpre će biti opisana konstrukcija (i modifikacija) konkretnih tokova podatka (aktivnih, ARP i reaktivnih), za koju se između ostalog koriste i pomenute vIP adrese. Najzad, biće izložena i apstraktna skica programskog rešenja za balansiranje opterećenja koja obuhvata dva softverska modula.

6.1 Konstrukcija i modifikacija tokova

Usled primene vIP adrese u tehničkom rešenju ovog istraživanja, bilo je neophodno vršiti i modifikacije tokova podataka. Metoda primenjena u ovom slučaju jeste NAT modifikacija paketa. Ona je podrazumevala prepisivanje izvornih i destinacionih IP adresa i TCP portova u zaglavljima paketa, kako bi se manipulisalo daljim tokovima podataka.

6.1.1 Rešenje za proaktivne tokove

Od trenutka kada se *Open vSwitch* (OVS) poveže sa SDN kontrolerom, SDN kontroler procesira sve nove OVS tokove. SDN kontroler mora obezbediti proaktivno ubacivanje tokova kako bi izmenio

pravila prosleđivanja saobraćaja i definisao koji od tokova će biti upravljan SDN kontrolerom, a koji tradicionalno. Konkretno, proaktivni tokovi su prikazani u tabeli 6.1.

	Tip paketa	Prosleđuje se SDN kontroleru	Tradicionalno procesiranje	Prioritet
P1	ARP	DA	DA	podrazumevan
P2	DstIP: vIP	DA	NE	1000
P3	Ostalo	NE	DA	500

Tabela 6.1 Proaktivni tokovi

Tok P1 odgovara ARP paketima koji se prosleđuju SDN kontroleru, a zatim obrađuju tradicionalno, dok im je prioritet podrazumevan. Toku P2 odgovaraju paketi kod kojih destinaciona IP adresa odgovara virtuelnoj IP adresi. Paketi ovog toka prosleđuju se SDN kontroleru i ne obrađuju se na tradicionalan način, dok im je prioritet veći u odnosu na ostale pakete. Konačno, ostali paketi se ne prosleđuju SDN kontroleru, već se samo obrađuju tradicionalno i prioritet im je niži u odnosu na pakete sa virtuelnom destinacionom IP adresom. Pod tradicionalnim obrađivanjem podrazumevaju se klasična L2 prosleđivanja i L3 procesi rutiranja u zavisnosti od funkcionalnosti samog SDN uređaja.

Proaktivni tokovi mogu definisati pravila koja menjaju podrazumevano ponašanje *bridge*-a SDN sviča. Dakle, ideja jeste da se SDN kontroleru šalju samo paketi sa destinacionom vIP adresom (P2) i ARP paketi (P1). Svi ostali paketi (P3) će biti automatski obrađeni tradicionalnim L2 prosleđivanjem na SDN sviču, bez posredstva SDN kontrolera, osim u slučaju pojave toka sa većim prioritetom.

6.1.2 Rešenje za ARP

U okviru predložene LBORU metode za balansiranja saobraćaja mora se razrešiti vIP u vMAC adresu (5.3 Nova šema balansiranja opterećenja - LBORU metoda). Server mora imati informaciju o destinacionoj MAC adresi pre slanja paketa vIP adresi. Zbog toga se šalje ARP zahtev za razrešavanje vIP u vMAC adresu. SDN kontroler mora takođe primiti taj isti ARP zahtev. Na taj zahtev, SDN kontroler obrazuje paket sa ARP odgovorom koji sadrži unapred definisano mapiranje vIP u vMAC adresu. SDN kontroler potom prosleđuje taj isti paket SDN sviču. Po prijemu paketa sa ARP odgovorom SDN svič prosleđuje ARP odgovor portu klijenta sa kojeg je ARP zahtev potekao. Nakon što klijent primi ARP odgovor za vIP adresu, u mogućnosti je da kao destinacionu adresu iskoristi vMAC adresu i pripremi paket za slanje.

SDN kontroler ignoriše bilo koji drugi ARP zahtev. Ostali ARP zahtevi se obrađuju tradicionalno, odnosno *flooding* metodom.

6.1.3 Rešenje za reaktivne tokove

Prvi paket veze poslat vIP adresi stiže na SDN svič, ali svič i dalje nema svoju instrukciju reaktivnog toka u tabeli tokova. Prema tome, SDN svič izvršava proaktivnu instrukciju P2, šalje paket SDN kontroleru, obavlja proceduru balansiranja opterećenja i na kraju odluči koji od servera (koja od rIP adresa) će biti zadužen za ovaj zahtev. SDN kontroler kreira dva reaktivna toka, prikazana u sledećoj tabeli 6.2.

	Tip paketa	Prosleđuje se SDN kontroleru	Tradicionalno procesiranje	Prioritet
R1	DstIP: vIP, SrcIP: srcIP, SrcTCPPort: srcTCPPort	NE	DA	1500
R2	SrcIP: rIP(srvN), DstIP: srcIP, DstTCPPort: srcTCPPort	NE	DA	1500

Tabela 6.2 Reaktivni tokovi

Oznake u tabeli i njihovo značenje:

- srcIP – rIP adresa klijenta
- srcTCPPort – port aplikacije klijenta (različit port za svaku novu TCP vezu)
- srvN – redni broj servera kome se zahtev prosleđuje (1 za WS1, 2 za WS2)
- rIP (srvN) – rIP adresa servera kome se zahtev prosleđuje
- rMAC (srvN) – rMAC adresa servera kome se zahtev prosleđuje

Reaktivni tok R1 predstavlja odgovor SDN kontrolera paketu zahteva koji dolazi sa sviča. Tok upućuje SDN svič na prepoznavanje paketa konkretne veze poslatog na vIP adresu i zatim ga modifikuje tako što zamenjuje destinacione adrese (vIP, vMAC) sa stvarnim adresama servera. Na te adrese se paket dalje prosleđuje primenom postojećeg L2 mehanizma. Definisanjem višeg prioriteta (1500) za instrukciju reaktivnog toka R1, u poređenju sa proaktivnim tokom P2, svaki budući paket sa istim parametrima (IP adresa izvora, port i vIP destinaciona adresa) biće prosleđen bez ikakvog posredovanja SDN kontrolera.

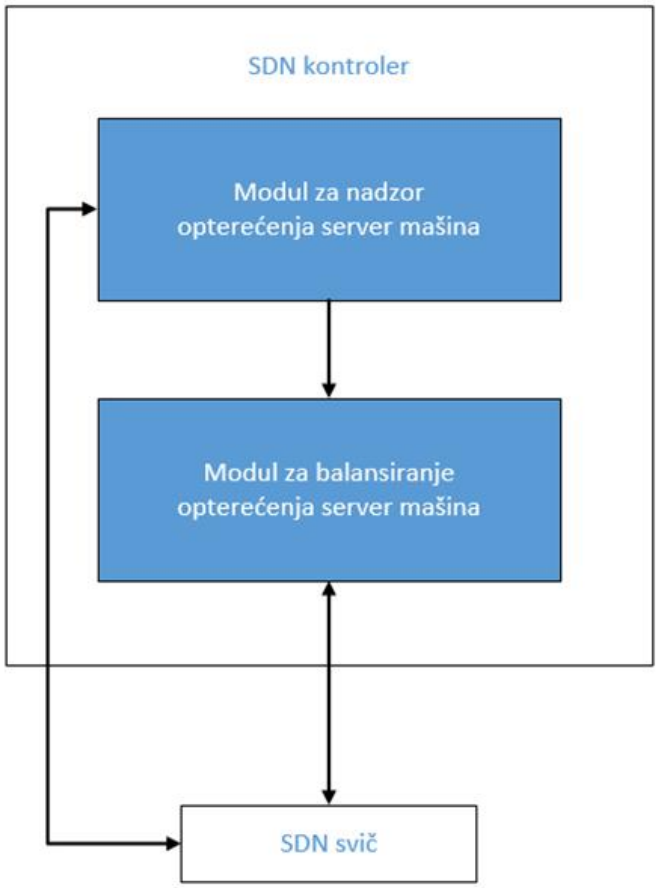
S obzirom na to da bi serverov odgovor morao biti vraćen pošiljaocu zahteva, jer klijent očekuje da vidi odgovor sa adrese na koju je poslao zahtev, SDN kontroler sa reaktivnim tokom R2 upućuje SDN svič na modifikovanje svakog paketa koji dolazi od posmatranog servera na taj način da rIP i rMAC budu zamenjene virtuelnim adresama (vIP i vMAC). Prioritet reaktivnog toka instrukcija R2

viši je od proaktivnog toka instrukcija P3, te će svič proslediti modifikovan paket na osnovu L2 mehanizma, bez posredstva SDN kontrolera.

6.2 Dizajn programskog rešenja

Na slici 6.1 prikazan je blok dijagram sa modulima programskog rešenja. Na SDN kontroleru su implementirana dva modula, jedan služi za praćenje opterećenja server mašina, a drugi je zadužen za samo balansiranje opterećenja više servera.

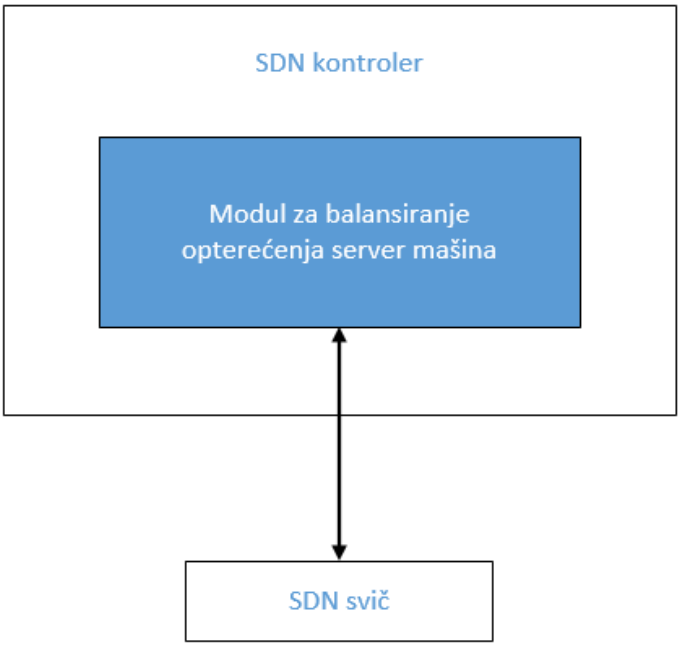
Northbound API, kojim se komunicira između kontrolera i aplikacije realizovan je implementacijom posebnog modula za nadzor opterećenja servera (SNMP protokol). *Southbound* API je realizovan primenom *OpenFlow* protokola u modulu za balansiranje opterećenja, tako što se ovim protokolom šalju direktive SDN sviču. Po ovim direktivama, SDN svič dalje preusmerava pakete saobraćaja.



Slika 6.1 Moduli programskog rešenja balansiranja opterećenja LBORU metodom

Ovakva podela rešenja na module je česta, kada su u pitanju metode dinamičkog balansiranja opterećenja. Metode dinamičkog balansiranja opterećenja, primenjene u testiranju rešenja ovog

istraživanja, nadziru opterećenje mašina kojima se korisnici opslužuju. Statičke metode balansiranja, s druge strane, ne poseduju modul za nadzor opterećenja mašina (slika 6.2).



Slika 6.2 Modul programskog rešenja statičkog balansiranja opterećenja

Kad je LBORU metoda u pitanju, modul za nadzor prikuplja podatke o parametrima opterećenja i proračunava ukupna opterećenja pojedinačnih servera (metrike), da bi kasnije modul za balansiranje iskoristio taj podatak kao ulaz u svoj program, čiji je zadatak da odluči o tome kojoj server mašini će pripasti opsluživanje datog korisnika. Modul za nadzor je potpuno nezavisan od modula za balansiranje. Nezavisnost se ogleda u činjenici da se programi ovih modula izvršavaju istovremeno, ne remeteći jedan drugom kontinualan rad u bilo kom trenutku. Ovo se ostvaruje upotrebom *thread*-ova (niti) pri izvršavanju programa ova dva nezavisna modula. Nit modula za nadzor periodično isporučuje podatak potreban za odluku o odabiru servera, modulu za balansiranje, pri čemu modul za balansiranje uopšte ne čeka da mu se novi podatak isporuči, već se u međuvremenu služi najsvežijim podatkom.

Na osnovu metrika modula za nadzor, modul za balansiranje prosleđuje korisnički zahtev onoj server mašini koja je najspremnija da opsluži korisnika (5.3 Nova šema balansiranja opterećenja - LBORU metoda).

Komunikacija SDN kontrolera sa svičem ostvarena je preko već ranije pominjanih *PacketIn* i *PacketOut* poruka (5.3 Nova šema balansiranja opterećenja - LBORU metoda). *PacketIn* poruke su one koje pristižu na kontroler, dok *PacketOut* poruke kontroler šalje sviču.

U prilogu disertacije (na kraju) detaljno su opisana programska rešenja oba modula. U toku izrade rešenja bilo je prvo potrebno implementirati modul za balansiranje, jer je on primenjen pri

ispitivanju rešenja ovog istraživanja (LBORU), ali i za implementaciju preostale dve metode balansiranja korišćene u uporednoj analizi: RR i LBBSRT. Posle toga, implementiran je modul za nadzor opterećenja, najpre za LBBSRT, a potom i za LBORU metodu. RR mehanizam ne poseduje modul za nadzor, jer je to statička metoda balansiranja (slika 6.2).

7. Testiranje predloženog rešenja

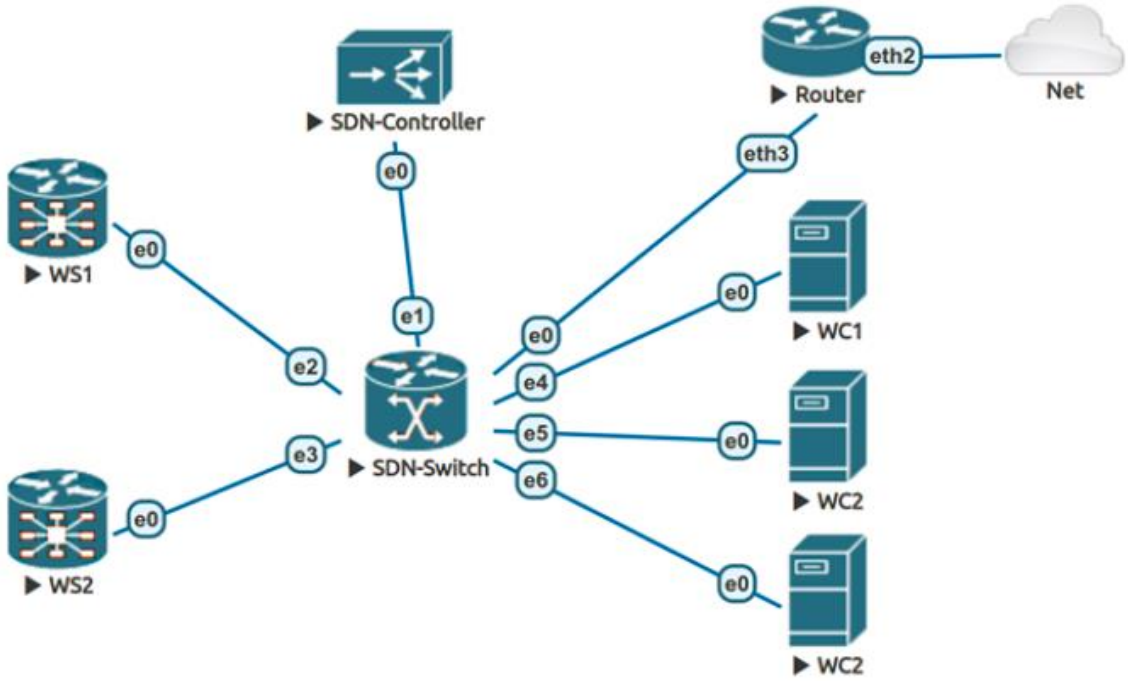
Za testiranje rešenja je bilo najpre potrebno postaviti okruženje za testiranje. Scenario za testiranje je definisan tako da se testno okruženje odražava na rad tradicionalno organizovane mreže sa *data* centrom. Šema za balansiranje opterećenja unutar hibridne SDN mreže na ovaj način dobija na praktičnoj vrednosti.

U procesu testiranja rešenja primenjena je metoda emulacije virtuelnog okruženja, dok je primenjeno programsko rešenje isto kao što bi bilo i na fizičkim mašinama.

7.1 Opis testnog okruženja

Za potrebe kreiranja testnog okruženja sa virtuelnim serverima i drugim mrežnim resursima primenjena je platforma *EVE-NG* [80]. Kreirane su sledeće virtuelne komponente (koje su prikazane na slici 7.1):

1. SDN svič – *Linux* OVS ruter sa ulogom SDN sviča,
2. Šest instanci virtuelnih mašina (eng. *Virtual Machine*, VM) sa *Linux Ubuntu* server operativnim sistemom:
    a) *web* server 1 (WS1), *web* server 2 (WS2) – VM instance *web* servera
    b) *web* klijent 1 (WC1), *web* klijent 2 (WC2), *web* klijent 3 (WC3) – VM instance *web* klijenta,
3. SDN kontroler – VM instanca sa *POX* SDN kontrolerom,
4. Ruter – instanca Mikrotik ROS rutera sa L3 funkcionalnostima i
5. Net – veza sa fizičkom tradicionalnom mrežom i internetom.



Slika 7.1 Testno okruženje na *EVE-NG* platformi

Većina današnjih servisa koje koriste kompanije su uglavnom internet servisi. Ove servise karakteriše visok nivo fleksibilnosti zasnovan na internet tehnologijama. Internet tehnologije omogućavaju dinamičko proširivanje resursa i podržavaju visok nivo performansi servisa. Neophodno je omogućiti dinamičko balansiranje saobraćaja između server mašina koje pružaju usluge istog servisa i čije resurse takođe treba dinamički proširivati. U ovom istraživanju su postavljena dva *web* servera u testno okruženje i implementirano je dinamičko raspoređivanje veza sa više klijenata.

Na slici 7.1 vidi se da su sve virtuelne mašine sa *Linux* operativnim sistemom povezane sa SDN svičem sa OVS platformom [81]. Ovaj svič upravlja tokovima paketa preko *OpenFlow* protokola. Implementiran je i ruter sa DHCP (eng. *Dynamic Host Configuration Protocol*) funkcionalnostima servera kako bi se omogućila dinamička i istrajna dodela IP adresa. Bilo je potrebno instalirati odgovarajući softver na virtualnim mašinama (primeri: *Java*, *Apache*, SNMP). Radi instalacije tih alata bilo je potrebno povezati virtualne mašine na internet preko rutera i Net interfejsa. Fizička mašina na kojoj je smešteno emulirano testno okruženje prosleđuje pakete kroz fizički interfejs povezan sa tradicionalnom mrežom.

OVS je konfigurisan na takav način da su svi SDN svič interfejsi u sastavu jednog virtuelnog SDN *bridge*-a (br-sdn). Kao entitet mreže, *bridge* dobavlja IP adresu zahvaljujući DHCP-u rutera. OVS se ponaša kao tradicionalan svič pre nego što se na SDN kontroleru podesi IP adresa. OVS

obavlja L2 proces prosleđivanja, popunjava tabele tokova na osnovu prvobitnih MAC adresa, prosleđuje frejmove na osnovu već zapisanih tokova i ukoliko je to neophodno, takođe distribuira frejmove. Od trenutka kada se IP adresa podesi na SDN kontroleru, SDN svič prosleđuje svaki paket SDN kontroleru na dalje procesiranje zbog nedostajućih instrukcija iz tabele tokova (ovo je podrazumevano ponašanje, a ne dizajniran tok). SDN kontroleru se prosleđuju samo ARP paketi i paketi čija je destinaciona IP adresa jednaka vIP adresi, a koje on potom procesira na način opisan u poglavlju 6.1.1.

Virtuelna mašina sa funkcionalnošću SDN kontrolera ima ulogu upravljanja u prethodno izloženom virtualnom okruženju za testiranje. Razni *OpenFlow* kontroleri su danas dostupni javnosti (na primer: *NOX, POX, Jaxon, OpenDaylight, Floodlight, IRIS, NodeFlow, Helios*). Ovo istraživanje se odlučilo za implementaciju *POX* kontrolera u *Python*-u. Osnovni razlozi ove odluke su jednostavnost implementacije i podrška mnogih biblioteka. Dakle, razvijen je modul koji preko *POX* kontrolera poseduje funkcionalnosti opisane u poglavlju koje predlaže novu šemu za balansiranje opterećenja saobraćaja.

7.2 Opis testova i procesa testiranja

Važnost ovog istraživanja ogleda se u postizanju preciznih rezultata testiranja predložene šeme za balansiranje opterećenja u hibridnoj SDN mreži. Bilo je neophodno obaviti istraživanje u skladu sa uslovima koji odgovaraju realnoj mreži u realnom okruženju. Uzimajući u obzir globalni porast internet servisa, efikasnost balansiranja opterećenja nove šeme testirana je na *web* serverima. Ceo mehanizam je testiran u emuliranom okruženju, sa virtuelnim mašinama kao klijentima, *web* serverima i SDN komponentama (SDN kontroler i SDN svič). Kako bi se postigli najpouzdaniji rezultati, testiranje je obavljano u kontrolisanim uslovima. Postepeno je povećavan broj konkurentnih korisnika kako bi se maksimizovalo opterećenje na *web* serverima i proverila efikasnost nekoliko mehanizama za balansiranje opterećenja, uključujući predloženu LBORU metodu. Za LBORU metodu je važno napomenuti da se nije zalazilo u dublju analizu ponašanja sistema pri izboru različitih vrednosti težinskih faktora k_i (iskaz (7)). Empirijskim putem, ustanovljene su sledeće vrednosti težinskih faktora, optimalne za potrebe testiranja LBORU metode:

1. procenat procesorskog vremena: 1
2. količina podataka pročitanih sa diska: 1
3. količina podataka upisanih na disk: 0.25
4. količina podataka preuzetih sa *Downlink*-a: 0.5
5. količina podataka poslatih na *Uplink*: 0.5

Razvijen je alat za testiranje koji šalje zahteve namenjene *web* serverima i sastoji se iz dve komponente:

1. komponenta posla – uspostavlja vezu sa *web* serverom, izvršava transakciju jednog zahteva i izveštava o vremenu potrebnom za njen završetak. Procedura ove komponente se izvršava u beskonačnoj petlji kako bi se simulirao jedan konkurentni korisnik, koji stalno obavlja transakciju;
2. komponenta konkurencije – povećava broj istovremenih korisnika u kontrolisanim uslovima, zapisuje vreme odziva svake transakcije, i obavlja statističke računske operacije.

Dok komponenta posla šalje zahteve *web* serverima, komponenta konkurencije prikuplja parametre testiranja kao što su:

- broj konkurentnih veza
- prosečno vreme potrebno za obavljanje jednog zahteva
- broj završenih transakcija u sekundi

Kako bi se izbegao uticaj mehanizama za optimizaciju resursa na komercijalnim *web* serverima (na primer: *Apache* ili *NginX*), koji smeta realnom uvidu u performanse predloženog rešenja, kreirani su „čisti" *web* serveri. Cilj je bio dobiti reprezentativne rezultate merenja koji se mogu primeniti u analizi nezavisno od konkretnog mehanizma optimizacije. Kod ponašanja servera je pisan u *Java* programskom jeziku i nezavisan je od bilo kakvih mehanizama optimizacije (svi alati razvijeni za potrebe testiranja mogu se naći na *Github*-u [82]). Testiranje je za cilj imalo da se opterete kreirani *Java* serveri sa ogromnim brojem korisničkih zahteva. Testiranje je počelo posmatranjem raznih servisa koji zahtevaju različite nivoe iskorišćenja serverskih resursa (procesor, RAM, mrežni resursi) tokom procesa generisanja odgovora. Drugim rečima, generisanje odgovora je proizvod opterećivanja *Java* servera u zavisnosti od zahteva.

Upoređeni su drugi često korišćeni mehanizmi balansiranja opterećenja, poput *Random, Round Robin* i LBBSRT metoda, kako bi se postigao jasan uvid u mogućnosti predloženog rešenja. U slučaju *Random* mehanizma, implementiranog direktno na nekoliko *web* klijenata, sa stanovišta servera zahtevi stižu kroz nasumičnu raspodelu (klijenti ne znaju koliko je koji server opterećen). Sa druge strane, okruženju je pridružen i SDN kontroler i njegovim posredstvom implementiran je *Round Robin*, a zatim i LBBSRT i LBORU mehanizmi, kako bi se stvorili uslovi za centralizovano upravljanje zahtevima i raspoređivanje veza ka *web* serverima.

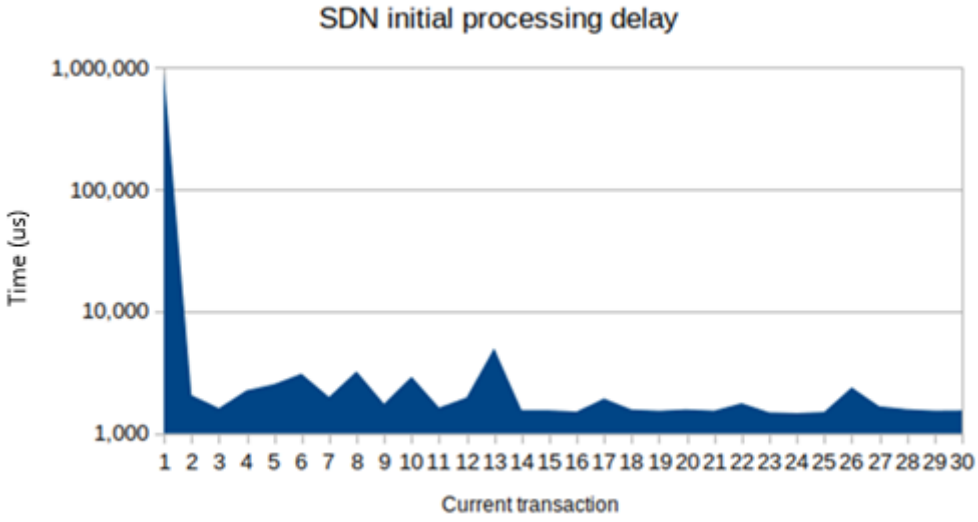
7.3 Rezultati testiranja i njihova analiza

Testiranje je sprovedeno u realističnom okruženju sa virtuelnim mašinama u *EVE-NG testbed* okruženju (šest ponavljanja). U rezultatima se javlja uticaj vremenskih varijacija opterećenja, koji ukazuje na to da bi dalje povećanje broja ponavljanja samo uvelo šum u rezultate.

U početnoj fazi testiranja, analiziran je uticaj obrade na svaki prvi paket i identifikovana je vrednost početnog kašnjenja obrade u SDN mreži. Nakon toga, analiziran je odnos između broja transakcija i konkurentnih klijenata (veza). Ova analiza teži da identifikuje kako početno kašnjenje SDN koncepta utiče na broj transakcija i srednje vreme transakcije. Na kraju je ocenjena efikasnost nove šeme za balansiranje i analizirana je neravnoteža opterećenja resursa na serverima, kao jedna od mera efikasnosti mehanizma za balansiranje opterećenja servera.

Kako bi dobijeni rezultati bili objektivno analizirani, bitno je uzeti u obzir prednost koju ima pristup direktne veze *web* klijenata sa *web* serverima naspram SDN koncepta. U pitanju je kašnjenje koje unosi obrada prvog paketa u okviru SDN veze. SDN svič šalje taj prvi paket SDN kontroleru na obradu, a potom SDN kontroler daje odluku (instrukciju) o daljem prosleđivanju. Dakle, direktne veze koje raspoređuje *Random* mehanizam za balansiranje opterećenja servera predstavljaju osnovnu orijentaciju u evaluaciji performansi.

Na slici 7.2 i tabelom 7.1 prikazan je uticaj obrade svakog TCP SYN paketa u SDN mreži (prvi paket svake nove veze). Početno kašnjenje usled obrade takvog paketa iznosi otprilike 1000 ms. Slika 7.2 i tabela 7.1 pokazuju rezultate merenja 30 transakcija iste veze. Izuzev prve, svaka naredna transakcija ima sličnu vrednost kašnjenja unetog obradom prvog paketa kao u slučaju direktne veze.

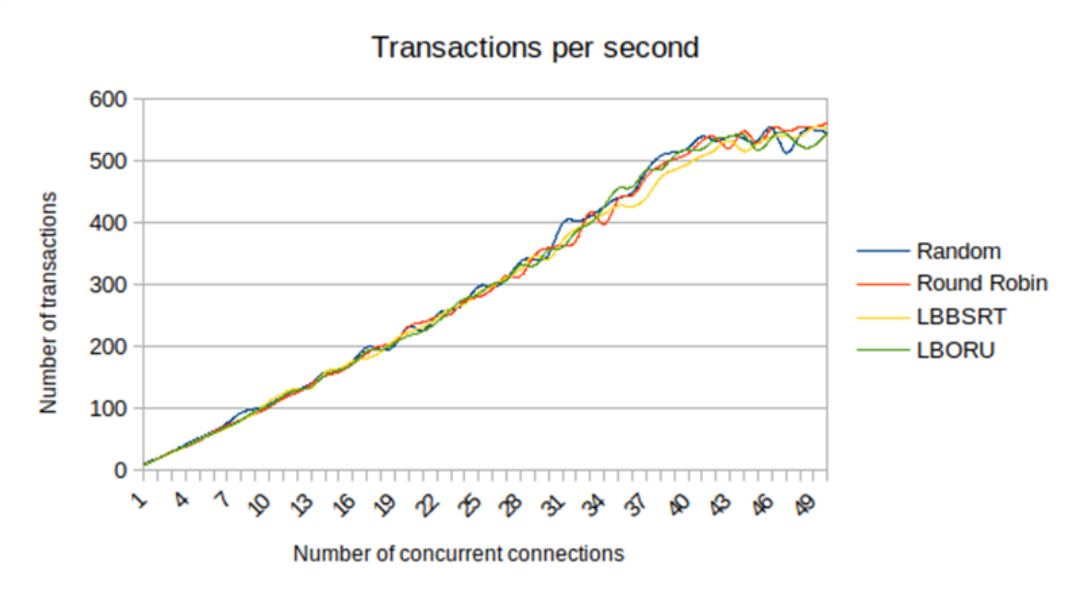


Slika 7.2 Početno kašnjenje uneto obradom prvog paketa SDN kontrolisane veze

Trenutna transakcija	Vreme [μs]	Trenutna transakcija	Vreme [μs]
1	905505.54	16	1505.52
2	2053.88	17	1926.61
3	1605.25	18	1566.68
4	2246.98	19	1529.63
5	2524.89	20	1569.66
6	3077.50	21	1532.17
7	1984.77	22	1765.31
8	3218.41	23	1485.58
9	1744.45	24	1462.71
10	2908.55	25	1495.92
11	1623.54	26	2377.01
12	1962.80	27	1665.47
13	4968.86	28	1575.27
14	1547.63	29	1533.74
15	1545.72	30	1538.86

Tabela 7.1 Početno kašnjenje uneto obradom prvog paketa SDN kontrolisane veze

Analizirani su rezultati dobijeni testiranjem prethodno opisanih mehanizama za balansiranje opterećenja servera (*Random*, RR, LBBSRT i LBORU) u okruženju opisanom u 7.1 Opis testnog okruženja. Jedan od ključnih parametara koji je analiziran jeste broj transakcija i njihova zavisnost od broja istovremenih korisnika (veza), kao što je prikazano na slici 7.3. Zaključak je da i pored kašnjenja uzrokovanog SDN konceptom, *Random* mehanizam nije imao značajnu prednost u pogledu broja transakcija u poređenju sa SDN rešenjima. Postoji linearna zavisnost između broja transakcija u sekundi i broja istovremenih veza – sve do 40 istovremenih veza. Dalji porast broja istovremenih veza uzrokuje gubitak zavisnosti jer postoji preopterećenost na oba servera. Kada se desi preopterećenje nema više značajne prednosti ni kod jednog mehanizma balansiranja.



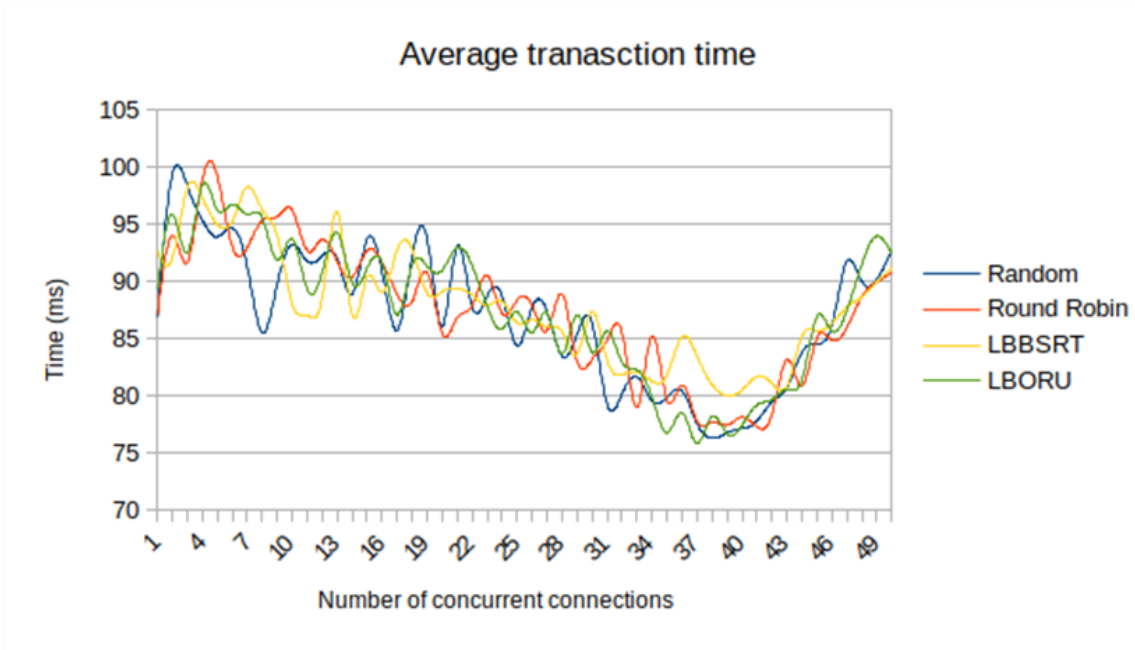
Slika 7.3 Poređenje broja transakcija u sekundi

Tabela 7.2 daje pregled poslednjih 20 vrednosti sa slike 7.3 za sva četiri mehanizma balansiranja opterećenja.

Br. istovremenih veza	**Prosečan broj transakcija u sekundi**			
	Random	**RR**	**LBBSRT**	**LBORU**
31	399.0	363.4	371.4	360.4
32	403.0	372.2	389.6	386.4
33	410.8	416.8	401.0	399.2
34	426.0	398.0	415.0	426.6
35	439.8	438.0	427.2	455.4
36	448.8	445.0	426.8	458.0
37	482.2	473.6	440.0	485.2
38	507.4	492.0	471.8	486.0
39	513.8	502.8	485.4	507.6
40	520.4	511.8	495.6	518.8
41	538.8	532.6	507.4	518.8
42	532.2	537.0	519.0	535.6
43	539.0	521.4	532.6	539.2
44	535.8	548.0	515.6	541.4
45	533.0	527.4	528.4	517.4
46	552.6	553.0	538.0	537.8
47	513.4	547.8	540.2	544.0
48	541.6	554.6	539.2	524.6
49	550.2	554.4	554.0	524.4
50	546.0	562.0	550.8	546.2

Tabela 7.2 Poređenje broja transakcija u sekundi

Analiza prosečnog vremena trajanja transakcije prikazana je na slici 7.4. Kašnjenje nastalo usled obrade prvog paketa veze ne predstavlja značajan uticaj.



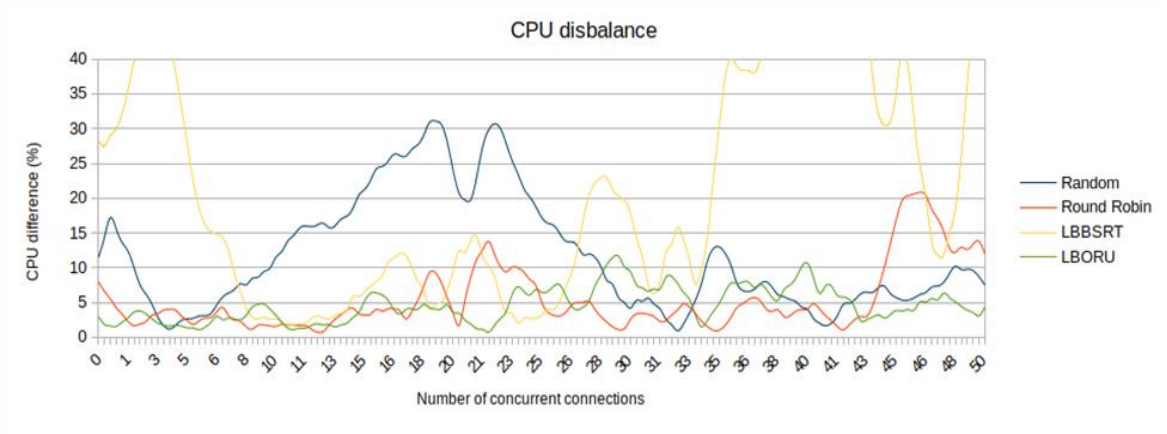
Slika 7.4 Poređenje prosečnih vremena trajanja transakcije

Tabela 7.3 daje pregled poslednjih 20 vrednosti sa slike 7.4 za sva četiri mehanizma balansiranja opterećenja.

Br. istovremenih veza	Prosečno vreme trajanja transakcije [ms]			
	Random	RR	LBBSRT	LBORU
31	79.30	84.87	83.15	85.69
32	80.16	85.54	81.86	82.95
33	81.67	79.00	82.09	82.27
34	79.57	85.26	81.29	79.89
35	79.88	79.65	81.83	76.76
36	80.43	80.95	85.18	78.52
37	77.61	77.84	83.51	75.88
38	76.31	77.70	80.99	78.23
39	76.78	77.47	80.02	76.71
40	77.14	78.14	80.51	77.43
41	77.82	77.32	81.69	79.22
42	79.49	78.38	81.12	79.66
43	80.78	83.18	80.89	80.59
44	83.79	80.91	85.18	81.42
45	84.55	85.07	85.69	86.95
46	86.20	84.84	86.33	85.68
47	91.78	85.96	87.77	87.47
48	89.98	88.70	88.67	91.53
49	90.18	89.95	89.95	93.98
50	92.79	90.81	91.20	92.35

Tabela 7.3 Poređenje prosečnih vremena trajanja transakcije

Kako bi se stekao uvid u efikasnost svakog od mehanizama za balansiranje opterećenja, analizirani su podaci koji opisuju opterećenje servera. Slika 7.5 prikazuje neuravnoteženost opterećenja procesora (CPU), tj. razliku između opterećenja procesora na mašinama WS1 i WS2. Ovo je jedan od ključnih pokazatelja efikasnosti mehanizama za balansiranje opterećenja. Manja neuravnoteženost predstavlja bolje raspoređeno opterećenje procesora na serverima.

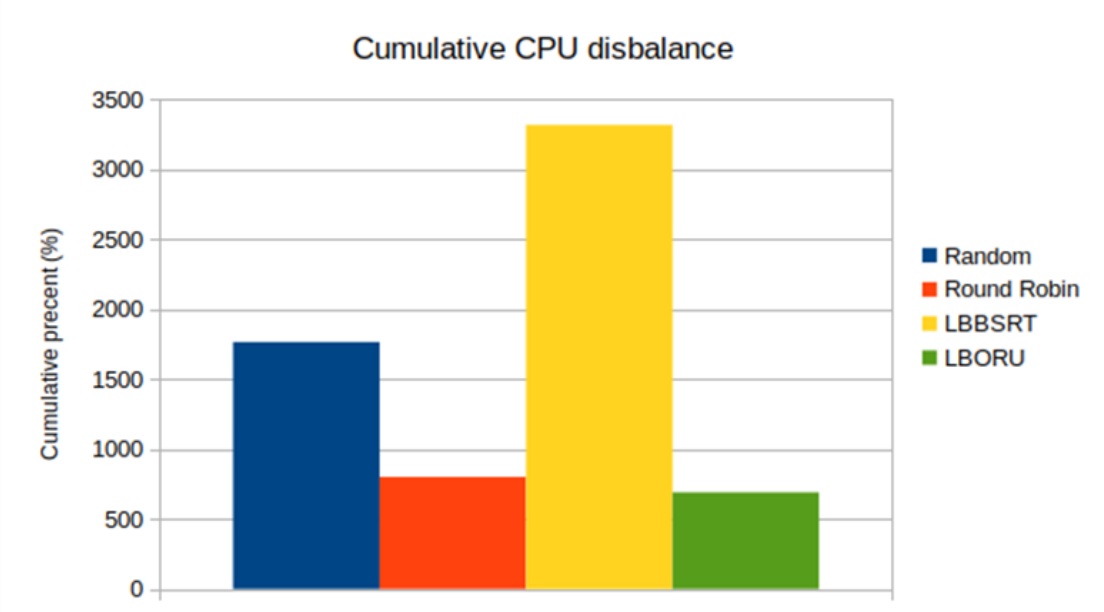


Slika 7.5 Poređenje neuravnoteženosti opterećenja procesora

Balansiranje opterećenja primenom *Random* mehanizma uzrokuje značajnu neuravnoteženost procesora, jer klijent nije svestan procesora kao serverskog resursa. Međutim, sa porastom broja istovremenih veza, dolazi do preopterećenja servera, a onda i neuravnoteženost postaje manje primetna. S druge strane, *Round Robin* mehanizam je pokazao zadovoljavajuće rezultate balansiranja opterećenja procesora do momenta kada se serveri ne preopterete. Usled nekonzistentnosti individualnih zahteva kada je u pitanju pristup CPU resursima, koji generišu odgovore na *web* serverima, primeti se da ovaj mehanizam stvara rastuću neuravnoteženost sa porastom broja istovremenih veza. Za merodavnu analizu rezultata dobijenih LBBSRT mehanizmom, bilo je potrebno uzeti u obzir da se on oslanja na RTT parametar. Za merenje RTT parametra preko ICMP protokola neophodno je zauzeti vrlo malo resursa na serveru, jer je odgovor na svaki serverov upit brz. Međutim, rezultati testiranja takođe pokazuju da LBBSRT ne može detektovati neravnotežu na dovoljno precizan način i vremenom izjednačiti opterećenje procesora na serverima.

Mehanizam predložen u ovom istraživanju – LBORU, nadgleda nekoliko parametara operativnog sistema server mašina. Odluka o prosleđivanju zahteva konkretnom serveru zasnovana je na parametrima dobavljanim u realnom vremenu, a preko SNMP protokola. Postignuti rezultati pokazuju da je maksimalna neuravnoteženost procesora oko 10 %, što se može smatrati vrlo malom vrednošću. Ono što je još važnije jeste da je vrednost neuravnoteženosti u tim granicama čak i posle preopterećenja servera.

Slika 7.6 pokazuje kumulativnu neuravnoteženost procesorskog opterećenja, ona predstavlja zbir neuravnoteženosti (procentualno) zasnovan na rezultatima sa slike 7.5. LBORU mehanizam daje najbolje rezultate u pogledu balansiranja procesorskog resursa na serverima. Istovremeno, *Round Robin* poseduje slične karakteristike, jer je najveći broj vrednosti (oko 75 %) dobijen u periodu pre preopterećenja server mašina.



Slika 7.6 Kumulativna neuravnoteženost opterećenja procesora

Kvantitativna analiza rezultata testiranja neuravnoteženosti opterećenja procesora data je Tabelom 7.4. Primenjene veličine pripadaju deskriptivnoj statistici i one su:

- Srednja vrednost:

$$\mu = \frac{\sum x_i}{N} \tag{9}$$

- Standardna devijacija:

$$\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}} \tag{10}$$

- Medijana:

$$Med(X) = \begin{Bmatrix} x_{N/2}, & ako\ je\ N\ parno \\ (x_{(N-1)/2} + x_{(N+1)/2})/2, & ako\ je\ N\ neparno \end{Bmatrix} \tag{11}$$

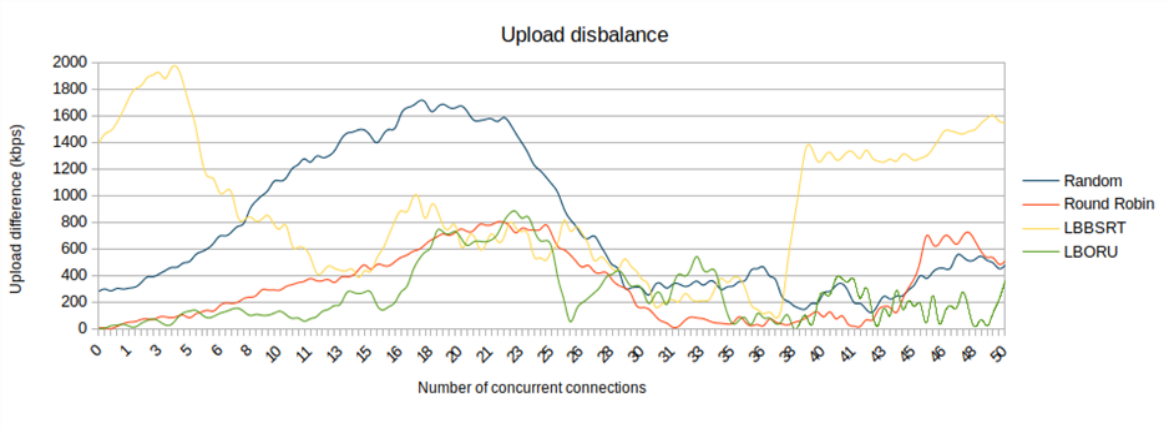
gde je $X = \{x_1, x_2, ..., x_N\}$ skup vrednosti prikazanih na slici 7.5, sortiran u rastućem redosledu. x_i je definisan kao $x_i \in X$ za svako $i = 1, 2,..., N$. Kardinalnost skupa N u ovom slučaju iznosi 150, jer postoji toliko izmerenih vrednosti (uzoraka) za svaku od testiranih metoda balansiranja opterećenja.

	CPU disbalance [%]			
	Random	**RR**	**LBBSRT**	**LBORU**
Srednja vrednost	11.75796	5.33520	22.09893	4.60107
Standardna devijacija	8.18145	4.64816	18.96571	2.49480
Medijana	8.92000	3.70000	14.40000	4.06000

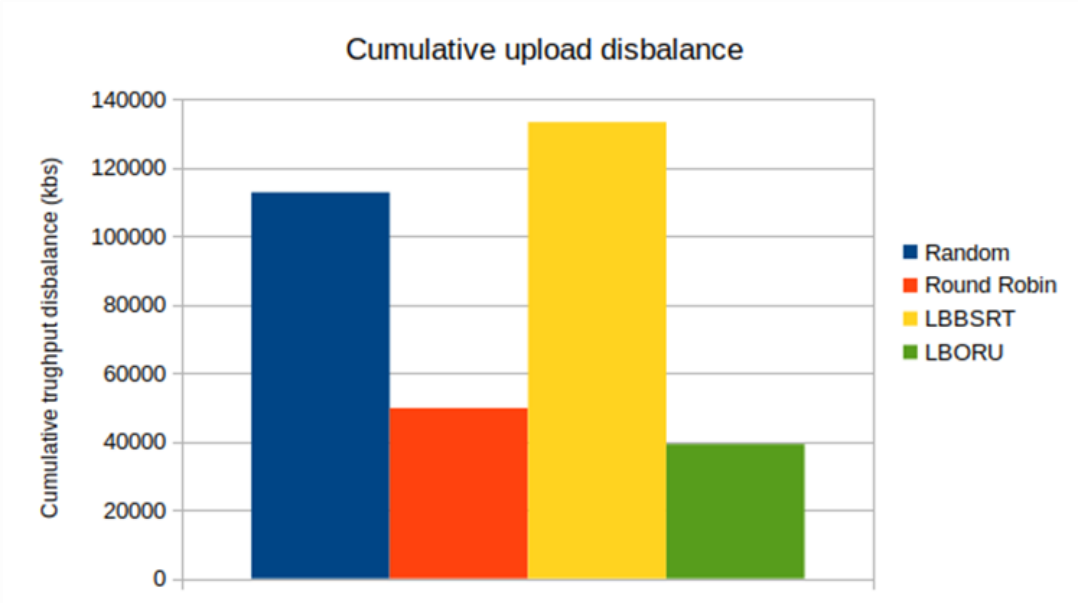
Tabela 7.4 Statistička poređenja neuravnoteženosti opterećenja procesora

U Tabeli 7.4 se zapaža da je prosečna opterećenost procesora za LBORU metodu najniža (4,6 %). Standardna devijacija opterećenosti procesora je takođe najniža u slučaju LBORU metode (2,49 %), što ukazuje na to da uzorci merenja opterećenosti procesora u proseku najmanje odstupaju od srednje vrednosti tog parametra opterećenja. Kada je vrednost medijane u pitanju, RR vrednost (3,7 %) je samo za nijansu manja od LBORU metode (4.06 %). Ona ukazuje na to da je jedna polovina izmerenih uzoraka manja ili jednaka od vrednosti medijane, a druga polovina veća ili jednaka vrednosti medijane.

Slike 7.7 i 7.8 pokazuju rezultate testiranja neuravnoteženosti količine poslatih podataka (*Link Upload*). Svaki odgovor zauzima različitu količinu ove vrste resursa, što procenu efikasnosti mehanizama za balansiranje čini donekle izazovnom.



Slika 7.7 Poređenje neuravnoteženosti količine poslatih podataka



Slika 7.8 Kumulativna neuravnoteženost količine poslatih podataka

Statističke mere neuravnoteženosti količine poslatih podataka prikazane su u tabeli 7.5. Za izračunavanje vrednosti ovih veličina primenjeni su isti matematički obrasci kao i kod merenja opterećenosti procesora.

	Upload disbalance [kbps]			
	Random	**RR**	**LBBSRT**	**LBORU**
Srednja vrednost	751.35067	331.64400	887.90373	261.40880
Standardna devijacija	506.43350	255.01116	486.79894	212.69859
Medijana	502.96000	299.88000	757.82000	184.70000

Tabela 7.5 Statistička poređenja neuravnoteženosti količine poslatih podataka

Testovi su pokazali da *Random* mehanizam raspoređivanja veza ka serverima daje solidne rezultate neuravnoteženosti procesora. Međutim, postoji značajna neuravnoteženost sa stanovišta mrežnih resursa, na šta ukazuje količina poslatih podataka. Nasuprot tome, *Round Robin* mehanizam implementiran na SDN kontroleru, iako nije svestan stanja na vezi (*linku*), uspeo je da obezbedi zadovoljavajuće balansiranje saobraćaja u mreži. Od LBBSRT mehanizma su se očekivali napredniji rezultati u pogledu balansiranja saobraćaja u mreži. Odstupanje od očekivanja je nastalo zbog opterećenja na mrežnom interfejsu i uticaja interfejsa reda čekanja na povećanje RTT vrednosti. SDN kontroler započne efikasnije balansiranje saobraćaja, međutim, dobijeni rezultati pokazuju da to nije dovoljno za neku značajniju promenu RTT vrednosti, te LBBSRT mehanizam ne donosi bolje odluke. Testovi su pokazali da LBORU mehanizam ispravno nadgleda trenutne vrednosti opterećenja

procesora i opterećenja mrežnih resursa, kao i da donosi najbolje odluke u raspoređivanju veza *web* klijenata ka serverima.

7.4 Diskusija rezultata testiranja

Pojava novih internet tehnologija izaziva nagli rast inteligentnih i sve zahtevnijih servisa, razvoj novih komunikacionih tehnologija i prisutnost mnogo više korisnika. Sve ovo zahteva definisanje efikasnije metode za balansiranje saobraćaja. Ovo istraživanje je analiziralo postojeće šeme balansiranja opterećenja u mreži i prepoznalo određena ograničenja. Šeme ne uključuju kontinualnu detekciju opterećenosti resursa u svoje procese odlučivanja. Osnovu rešenja ovog istraživanja čine upravo ta kontinualna detekcija opterećenosti i implementacija mehanizma koji omogućava njeno učešće u svom procesu odlučivanja. Mehanizam takođe dozvoljava prilagođavanje šeme za balansiranje saobraćaja zahtevima konkretnog servisa (po pitanju resursa fizičke arhitekture).

Rezultati testiranja ukazuju na uticaj obrade svakog prvog paketa veze, i otkrivaju vrednost početnog kašnjenja u SDN mreži. Potom se daljom analizom odnosa između broja transakcija i istovremenih veza ustanovilo da ovo početno kašnjenje, nastalo uvođenjem SDN koncepta, nije značajno uticalo na broj obavljenih transakcija i prosečno vreme obavljanja jedne transakcije. Dakle, postoji linearna zavisnost između broja transakcija u sekundi i broja istovremenih veza. Ocenjivanje efikasnosti predložene LBORU šeme analizirano je kroz posmatranje neravnoteže korišćenosti resursa na serverima kao jednog od pokazatelja efikasnosti mehanizama za balansiranje opterećenja na serverima. Analize potvrđuju da povećanjem broja istovremenih veza i opterećenja na serverima, neravnoteža postaje značajno manje primetljiva.

Radi jednostavnijeg poređenja rezultata testiranja, izmerena je kumulativna neuravnoteženost procesora. Rezultati dobijeni LBORU mehanizmom su vidljivo najbolji u pogledu ravnopravne opterećenosti procesora na serverima. Pored toga, predstavljeni su i rezultati merenja kumulativne neuravnoteženosti količine podataka na *Uplink*-u, gde je ponovo LBORU mehanizam imao najbolje performanse. Dakle, testovi su pokazali da LBORU mehanizam, koji nadzire opterećenost procesora i mrežnih resursa (najbitnijih za sprovedeno testiranje), donosi bolje odluke pri balansiranju opterećenja od ostalih testiranih mehanizama.

8. Primena modela hibridnog SDN-a u 6G mrežama

Akademska i stručna zajednica su krajem prošle decenije započele aktivnosti na razvoju i implementaciji nove 6G mobilne mreže. Definisani su ključni pokazatelji performansi (eng. *Key Performance Indicators,* KPI) budućih 6G mreža i sagledane mogućnosti dalje automatizacije upravljanja mrežnom infrastrukturom i primene znatno većeg stepena veštačke inteligencije. Uloženi su veliki napori, da se u okviru pristupnih mreža, ali i u *edge*-u mreže implementiraju naprednija rešenja. Međutim, praksa je pokazala da to nije dovoljno, jer da bi se implementirali sve zahtevniji servisi (pre svega u pogledu potrebnog nivoa QoS-a) i time postigao željeni napredak u društvu, neophodno je obezbediti još programabilnije okruženje u okviru *core*-a mreže. Drugim rečima, neophodno je postići veći nivo fleksibilnosti i skalabilnosti u okviru infrastrukture *core*-a.

Polazeći od ove činjenice, u okviru istraživanja su analizirana postojeća rešenja u infrastrukturi 5G mreža, koja koriste unapred definisane redove čekanja (eng. *queue*). Takva rešenja su u suprotnosti sa ključnom intencijom u razvoju mreža, koja podrazumeva obezbeđivanje značajno većeg niova programabilnosti u mreži. Zato je cilj ovog istraživanja bio, da se u domenu upravljanja redovima, razvije novo rešenje koje bi se zasnivalo na primeni softvera za dinamičko upravljanje redovima i time proces upravljanja redovima u potpunosti premestio na sloj aplikacije (smanjila zavisnost od fizičke mrežne infrastrukture). Ovakvo rešenje se može efikasno primeniti i na 5G mreže kako bi se rešili složeni zahtevi servisa.

Ovde se mora imati u vidu, da je razvoj 5G mreža doveo do značajno veće primene softverskih i rešenja zasnovanih na nekoj od tehnika virtuelizacije [83]. Međutim, uprkos ovoj činjenici, praksa je pokazala da 5G mreže još uvek ne uspevaju da pruže zadovoljavajuća rešenja za jedan broj servisa kao što je npr. multi-senzorska holografska teleportacija, gde se zahteva brzina prenosa podataka u Tbps i ms kašnjenja [84]. Upravo imajući ovo u vidu, naučna zajednica je definisala ključne pokazatelje performansi za 6G ekosistem [85] i ukazala na potrebu da se isti pažljivo mere radi

tranzicije sa *Network-as-an-Infrastructure* (NaaI) koncepta na *Network-as-a-Service* (NaaS) koncept [86]. Ova tranzicija podrazumeva da se ubrza prelazak na rad u *cloud* okruženju, sa primenom tehnologije *cloud/edge* računarstva, što u krajnjoj instanci vodi do konvergencije umrežavanja i *cloud/edge* računarstva [87].

Drugim rečima, buduće 6G mreže treba da poseduju E2E arhitekturu (eng. *end-to-end*) zasnovanu na servisima, u kojoj softverski definisano umrežavanje (SDN), virtuelizacija mrežnih funkcija (NFV) i podela mreža na slajseve igraju fundamentalnu ulogu [88][89]. Ove tehnologije obezbeđuju značajno veći nivo programabilnosti u mrežnoj infrastrukturi i omogućavaju stvaranje više logičkih mreža na jednoj zajedničkoj, odnosno deljenoj fizičkoj infrastrukturi. Primenom rešenja zasnovanih na ovim tehnnologijama, očekuje se realizacija većih brzina prenosa podataka, poboljšana spektralna i energetska efikasnost, bolja pokrivenost, širok propusni opseg, izuzetno visoka pouzdanost, veoma malo kašnjenje i dinamičko upravljanje QoS-om u svim segmentima mreže što je od suštinskog značaja za postizanje visoke fleksibilnosti u korišćenju dostupnih mrežnih resursa.

Danas, u većini SDN implementacija, potreban QoS se obezbeđuje statički, rezervacijom resursa, prema unapred definisanim pravilima, pa se npr. paketi mapiraju u postojeće redove čekanja sa različito dodeljenim prioritetima [90]. Nedostatak dinamičkog upravljanja redovima (tj. kreiranje, modifikacija i uklanjanje redova) je očigledan, pa se ovaj deo istraživanja fokusira na razvoj i implementaciju prihvatljive metode za dinamičko upravljanje QoS-om. Drugim rečima, potrebno je definisati *framework* za izradu *core*-a 5G/6G mreža, uključujući nov pristup upravljanju servisima. To podrazumeva vezivanje pojedinačnih servisa u grupe, koje najverovatnije nudi isti provajder servisa. Osnov budućeg rešenja je ideja, da se svaka grupa servisa upravlja sopstvenim autoritativnim kontrolerom i da se kroz dinamičko upravljanje redovima obezbedi dinamički QoS. Centralni deo rešenja je dinamička konfiguracija parametara redova u skladu sa zahtevima navedenim kroz upravljanje servisima na sloju aplikacije. Konkretno, primenjen je OVSDB protokol [91], kako bi se proširila funkcionalnost SDN kontrolera (pored funkcija koje podržava *OpenFlow* protokol [92]). Ovaj pristup uvodi znatno veći stepen programabilnosti fizičke i virtuelne infrastrukture. Sa naučnog i metodološkog stanovišta, primena ovakvog rešenja treba da omogući:

- Optimizovan E2E *slicing* u mrežama sa deljenom infrastrukturom kroz više domena;
- Dinamičko upravljanje QoS-a u SDN okruženju kroz dinamičko upravljanje redovima čekanja.

Dakle, jasno je da će buduće 5G/6G mreže biti hibridni SDN sistemi, koji obuhvataju složene arhitekture bežičnih i optičkih komunikacija. Uvođenje novih tehnologija nije dovoljno za rešavanje svih izazova u okviru postojeće arhitekture, ako se ne osmisle koncepti arhitekture kojima bi se realizovala robusna, fleksibilna, isplativa i inteligentna mreža [93]. Sigurno je, da SDN i NFV

tehnologije predstavljaju osnovu u izgradnji takvih arhitektura, gde se potrebna fleksibilnost obezbeđuje razdvajanjem mrežnih funkcija od hardvera i njihovom implementacijom u softveru. Ovime se stvaraju uslovi za efikasnu implementaciju i brže predstavljanje novih i složenijih servisa tržištu [94], a samim time i omogućava brz rast broja korisnika, pojava novih vidova komunikacije i omogućava *Over-The-Top* (OTT) igračima da povežu svoje servise i aplikacije sa 5G/6G infrastrukturom [95].

Jedan od ciljeva ovog istraživanja jeste i da se ispita mogućnost poboljšavanja QoS performansi SDN mreža i to u okolnostima kada se broj servisa povećava i menja količina korišćenih mrežnih resursa. Jasno je, da u takvim okolnostima SDN kontroler ima ključnu ulogu kako bi se na fleksibilan način zadovoljili zahtevi različitih servisa (SDN kontroler poseduje mogućnost kontrole svakog toka paketa podataka). Međutim, stvarnost je da neke funkcionalnosti koje bi trebalo da obezbede fleksibilniju konfiguraciju u SDN mreži još uvek nedostaju. Na primer, postojeći *OpenFlow* svičevi dozvoljavaju samo obradu paketa *Match-Action* metodom (ova metoda koristi fiksni skup polja paketa i ograničenu specifikaciju *OpenFlow* protokola, sa ograničenim skupom akcija obrade paketa [96]).

8.1 Postojeća istraživanja u ovoj oblasti

Paralelno sa identifikacijom različitih QoS zahteva, neophodno je razviti nova rešenja u domenu *network slicing*-a. Ta rešenja treba da omoguće provajderu da izvrši prilagođavanje i optimizaciju slajsova kao virtuelnih mreža u skladu sa servisnim zahtevima i politikom kojom se definišu pojedini slajsovi. Međutim, savremeni trendovi u oblasti IKT-a vode ka porastu broja korisnika, ali i servisa koji će biti pružani preko deljene infrastrukture. Stoga se fokus istraživanja u ovoj oblasti pomera ka pronalaženju optimalnog kriterijuma za kreiranje slajsova i dinamičko upravljanje QoS-om preko deljene SDN infrastrukture (fizičke i virtuelne infrastrukture).

Slajsevi se kreiraju primenom NFV tehnologije i dinamički distribuiraju širom mreže obuhvatajući više administrativnih domena. Sama primena NFV tehnologije ima za cilj da se kontrolna funkcija odvoji od ravni podataka i putem primene SDN funkcionalnosti centralizuje u jednoj tački odakle bi se vršila konfiguracija i upravljanje mrežnom infrastrukturom. Važno je istaći, da se u deljenim 5G/6G mreža primenjuje tzv. *multi-tenancy* arhitektura softvera, u kojoj više nezavisnih instanci jedne ili više aplikacija radi u zajedničkom fizičkom okruženju. Instance (eng. *tenants*) su logički izolovane unutar fizičke infrastrukture. Zahtevan QoS servisa može se garantovati izolovanjem resursa koji pripadaju različitim slajsevima, što je od vitalnog značaja za pouzdan rad 5G/6G sistema [97]. U zajedničkoj 5G/6G mreži, tj. u *multi-tenancy* okruženju, kreiranjem slajsova

se omogućava različitim korisnicima, servisima, ali i provajderima servisa da dele istu infrastrukturu i da grade potpuno odvojene E2E mreže [85][98]. Ovakvo okruženje donosi ekonomske benefite za provajdere servisa, jer smanjuje potrebu za ulaganja u zasebnu infrastrukturu.

Realizacijom *multi-tenancy* okruženja odvaja se uloga provajdera infrastrukture od uloge provajdera servisa i stvara tržišno okruženje koje povećava konkurenciju na obe strane. Što se tiče bezbednosti, korišćenje zajedničke infrastrukture podrazumeva obaveznu implementaciju onih rešenja koja treba da omoguće maksimalnu izolaciju saobraćaja koji pripada različitim slajsovima [99]. Dakle, imajući u vidu da se softverski definiše struktura slajsa, stvaraju se uslovi da se heterogeni infrastrukturni resursi iskoriste na racionalan i isplativ način [100][101]. Zato je politika definisanja slajsova od ogromnog značaja i zaokupljuje veliku pažnju akademske zajednice. Neki istraživači [102] predlažu primenu tzv. *mixed-integer* linearnog programa koji minimizuje kašnjenje prema zahtevima klijenta i ograničenja propusnog opsega servera, na osnovu niskog i srednjeg intenziteta saobraćaja. Sa ovim ciljevima, ispostavilo se da koncentrisanje saobraćaja na najbližem serveru daje optimalno rešenje u NFV kontekstu.

Puštanje u rad inteligentnih i adaptibilnih okruženja/aplikacija pokrenulo je evoluciju 5G/6G mreža. U pojedinim istraživanjima [103], autori naglašavaju da 6G mreža treba da obezbedi manje kašnjenje i veću pouzdanost, omogućavajući veoma masovne M2M komunikacije i primenu komunikacija male snage. Autori u [104] pretpostavljaju da se dinamička konfiguracija mreža zasniva na trenutnoj upotrebi mreže. Uspostavljanjem efikasne šeme za deljenje mreže, više korisnika, koji mogu posedovati konfliktne zahteve za resursima, dobijaju pristup različitim delovima ograničenih resursa.

Upravo, proces implementacije inteligentnih servisa u 5G i 6G multiservisnim okruženjima privlači posebnu pažnju [105]. Autori osmišljavaju taksonomiju za kreiranje slajsova u mreži pomoću različitih parametara (npr. fundamentalni principi dizajna, ulančavanje servisnih funkcija, fizička infrastruktura i bezbednost) i shodno tome predlažu nekoliko kriterijuma za kreiranje slajsova u mreži i u skladu sa time moguća rešenja.

Zato je potrebno da postoje studije [106] koje analiziraju nove trendove i izazove u kreiranju slajsova, 3GPP procesu standardizacije i s tim u vezi povezane mehanizme. Međutim, primećuju se određeni nedostaci, na primer, složenost 3GPP rešenja, nedostatak portala korisnika i odvojena orkestracija mrežnih slajsova i njihovih servisa. U postojećim rešenjima koja se odnose *na network slicing* u 5G mrežama [107][108] postoje jasno izražena ograničenja koja se naročito odnose na upravljanje i orkestraciju. Primećuju da operacije na nivou slajsova nisu dobro odvojene od drugih procesa, što rezultira složenim interakcijama između 5G mrežnih komponenti u celokupnoj arhitekturi mrežnih slajsova.

U procesu traženja odgovarajućeg rešenja za ove probleme, neki istraživači [107] predlažu modularni 6G-LEGO model rada mreže. Kod ovog modela, slajsevima se upravlja i orkestrira primenom namenskih računarskih sistema, a slajsevi sa višestrukim domenima se oslanjaju na višestruke agnostičke orkestratore. U [108], autori se zalažu za ono što nazivaju nultim dodirom 6G masivnog slajsovanja mreže. Uvode novi analitički *engine* za predviđanje preraspodele resursa, učenjem van mreže uz poštovanje nekih ograničenja koja se tiču SLA sporazuma na nivou servisa.

Očigledno je da se danas velika pažnja posvećuje definisanju optimalnog modela za kreiranje slajsova. Pojedini autori [109] razmatraju mogućnosti za kreiranje slajsova na tri različita sloja (infrastruktura, mrežni i servisni sloj) i izlažu pregled postojećih arhitektura *network slicing*-a. U [87][110], autori opisuju glavni cilj 5GEx projekta da se decentralizovani kaskadni pristup zaduži za orkestraciju servisa između provajdera. Kombinovanjem mrežnih funkcija infrastruktura kao usluga, virtuelnih mrežnih funkcija kao usluga i usluge povezivanja, 5GEx uvodi *Slice-as-a-Service* (SlaaS) paradigmu. Prema ovom pristupu, svaki provajder deluje kao preprodavac servisa kupaca, a isporučeni servisi mogu sadržati podservise i/ili resurse drugih provajdera.

Sveobuhvatan pregled tehnologija i osnovnih principa izgradnje slajseva mrežne arhitekture [111] ukazuje na zahteve koje je potrebno zadovoljiti u *core*-u mreže kako bi se realizovalo kreiranje slajsova mreže i izazovi u vezi sa alokacijom resursa slajsa. Autori u [112] predlažu nove tehnologije, kao što su vazdušni interfejs i transmisione tehnologije i nove arhitekture mreže. Oni predlažu *network slicing* kako bi se prevazišli nedostaci 5G mreže, zajedno sa arhitekturom bez ćelija i računarstvom u *cloud*-u. Jedno od suštinskih karakteristika buduće mreže biće izuzetna fleksibilnost u izgradnji i konfiguraciji više od 10 miliona privatnih mreža i mnoge automatizovane i napredne metode kreiranja slajsova u mreži. U [113], autori očekuju da vide do 10.000 slajsova kod provajdera servisa, kojim mreža pametno deli virtuelne resurse.

Akademske i stručne zajednice sprovele su značajnu količinu istraživana u vezi sa QoS-om u SDN mrežama sa *OpenFlow* protokolom. Razvoj 5G i 6G mrežne arhitekture podstiče učestalost istraživanja na ovu temu, fokusirajući se na obezbeđivanje adaptivnog QoS-a u SDN okruženju, sa raznolikošću svičeva, E2E kvalitetom iskustva (QoE) i dinamičkim upravljanjem redovima [114].

U [115], autori predstavljaju rešenje za obezbeđivanje E2E QoS-a. Oni implementiraju QoS kontroler koji usmerava tokove saobraćaja i dodeljuje odgovarajuće resurse duž saobraćajnih putanja prema potrebnom nivou performansi i garancijama koje su potrebne za različite primene. Fokus je na aspektima ravni upravljanja QoS-a, a ne na uticaju koji ovi mehanizmi imaju kada se primenjuju dinamički.

U svojim istraživanjima, neki autori naglašavaju različitost svičovanja i SDN operacija u upravljačkoj ravni. Tako npr. u [116–119], autori analiziraju skalabilnost i performanse sviča u

pogledu sve većeg broja pravila i saobraćajnih prioriteta. Oni takođe uzimaju u obzir uticaj nekih anomalija upravljačke ravni.

Studije [120][121] ispituju funkciju ograničenja brzine podataka *OpenFlow* protokola za kombinaciju saobraćajnih tokova „slonova“ i „miševa“, koji se redovno pojavljuju u proizvodnim mrežama. Istraživanje izloženo u [120] obavljeno je na statičkom QoS podešavanju u *Mininet* okruženju, dok su u [121] autori koristili svič *Pica8 P3290*.

U [122], autori koriste duboku inspekciju paketa kako bi omogućili QoS mehanizam prepoznavanja aplikacija za E2E sisteme inženjeringa saobraćaja u SDN mrežama. Implementirani sistem povećava propusnost i smanjuje kašnjenje distribucijom tokova na više redova sa različitim prioritetima, pod kontrolom administratora mreže. Adaptivni QoS mehanizam može biti od pomoći u prvoj fazi izgradnje SDN okruženja u hibridnom SDN-u, gde *OpenFlow* i tradicionalne mreže postoje zajedno [123].

Neki autori u svojim istraživanjima posvećuju posebnu pažnju izazovima koji se javljaju u SDN mrežama koje pružaju instant servise. Njihovo istraživanje postavlja za cilj razvoj mehanizma za upravljanje propusnim opsegom i primenu odgovarajuće politike ograničenja brzine prenosa prilikom zagušenja saobraćaja, za svaku aplikaciju. Na primer, oni razvijaju mehanizam za dinamičko rutiranje u SDN mreži na osnovu statistike praćene u realnom vremenu i izračunavanje optimalnih ruta sa minimalnim troškovima [124]. Oni integrišu ovaj mehanizam sa određenim algoritmom mašinskog učenja koji omogućava klasifikaciju toka i identifikaciju aplikacije.

Zagušenje veze je ozbiljan problem koji značajno utiče na performanse uređaja i celu mrežu. U [125], autori analiziraju ovaj problem u kontekstu prioritetizacije kašnjenja i gubitka paketa na ruterima. Oni predlažu *Weighted Queue Dynamic Active Queue Management* zasnovan na dinamičkom praćenju i reakciji, koje definišu težinski faktori i pragovi redova, dinamički prilagođeni na osnovu opterećenja saobraćaja.

Nedavno, autori su se fokusirali na integraciju SDN arhitekture sa konceptom servisno-orijentisane arhitekture (SOA) [126]. Ovaj koncept arhitekture donosi izazov održavanja QoS-a u mrežama. Autori zaključuju da i dalje postoje nedostaci u razvoju i primeni upravljanja QoS-om u SDN mrežama zasnovanim na SOA. Oni kategorizuju upravljanje QoS-om u pet glavnih kategorija i analiziraju uticaj kategorija na obezbeđivanje garantovanog E2E QoS.

Dinamičko upravljanje QoS-om će postajati sve značajnije za 5G i posebno 6G mobilne komunikacije, i zahtevaće brze i efikasne odgovore na mnoge izazove kao što su manja kašnjenja, veći obim saobraćaja i brzine prenosa podataka. U [127], autori predlažu rešenja u okviru sinergije tehnologija, kao što su NFV i SDN tehnike, preko radio sistema koji podržavaju *cloud*. Oni ukazuju na potencijalne prednosti kao što su omogućavanje udruživanja resursa, skalabilnost, slojevito međusobno povezivanje i spektralna efikasnost.

Sa aspekta velikih mreža ili mreža nosioca zasnovanih na SDN tehnologiji, kao npr. 5G i 6G mreža, specifične komponente *OpenFlow* arhitekture će biti predmet kontinuiranog razvoja. Promene u okviru *OpenFlow* standarda, koje se odnose na QoS, predstavljaju značajan izazov jer u okviru *OpenFlow* protokola još uvek postoji mnogo slobodnog prostora za implementaciju novih, sveobuhvatnijih rešenja, bogatih funkcijama. Kako bi izveli QoS eksperimente, autori u [128] uvode proširenje arhitekture za *OpenFlow* okruženje. Dalje, autori identifikuju trenutna ograničenja QoS-a na *OpenFlow* sviču, predlažu i implementiraju poboljšanja u vezi sa konfiguracijom redova, i proširuju *OpenFlow* protokol definisanjem podešavanja reda na osnovu određenih tipova redova. Neki autori su se fokusirali na aspekte ravni upravljanja i kontrole, kako bi postigli centralizovani E2E QoS [129][130]. Bez proširenja *OpenFlow* specifikacije, oni su odlučili da implementiraju *framework* za upravljanje QoS politikom koji obezbeđuje interfejs za određivanje SLA zasnovanih na QoS-u i nameće upotrebu *OpenFlow* API-ja.

Koristeći GI/M/1/K i M/M/1 redove iz teorije redova, autori u [131] modeluju svič i kontroler. Oni upoređuju dve različite implementacije svičeva (deljeni bafer i dva prioritetna bafera) i zaključuju da bi prioritetno baferovanje trebalo da bude preferirani mehanizam za mobilne mreže.

Može se zaključiti da se većina istraživanja fokusira na pristupne mreže i *edge* računarstvo. Imajući u vidu heterogenost budućih mreža, ističe se da je neophodno detaljno razmotriti potencijalne izazove u *core*-u budućih mreža. Da bi se postigli osnovni ciljevi 5G/6G mreža, mora se podsticati veći kapacitet sistema, veća brzina prenosa podataka, smanjiti kašnjenje i poboljšati kvalitet usluge (QoS). Postoje studije koje se bave implementacijom dinamičkog QoS-a u 5G/6G mrežama. Međutim, one se ne bave pitanjem dinamičkog upravljanja redovima (tj. kreiranjem redova i njihovom modifikacijom i uklanjanjem).

Oslanjajući se isključivo na unapred definisane redove, nijedno dosadašnje istraživanje ne predviđa mogućnost dinamičnog stvaranja QoS redova čekanja [132]. Što se tiče QoS-a, takav pristup je ograničavajući faktor fleksibilnosti sistema. Ideja primene dinamičkog upravljanja redovima jeste da se u značajnoj meri obezbedi neophodna fleksibilnost u *core*-u mreže. Zato je fokus ovog istraživanja, upravo na razvoju i implementaciji SDN rešenja koje bi dinamički obezbedilo potreban nivo QoS-a u *core*-u mreže.

Fleksibilnost i skalabilnost treba da teže ka zajedničkom cilju [133], te ovo istraživanje predlaže kreiranje slajsova u mreži, u kombinaciji sa dinamičkim upravljanjem redovima, kao vid rešenja za raznolike zahteve servisa. U tom smislu, primenjeno je odgovarajuće rešenje za obezbeđivanje bolje organizacije slajsova u okviru postojećeg *multi-slice* okruženja. Ova ideja o organizaciji slajsova treba da predstavi grupu sličnih servisa kao jedan slajs. Takav slajs se može sastojati od servisa različitih provajdera koji pripadaju istoj grupi servisa.

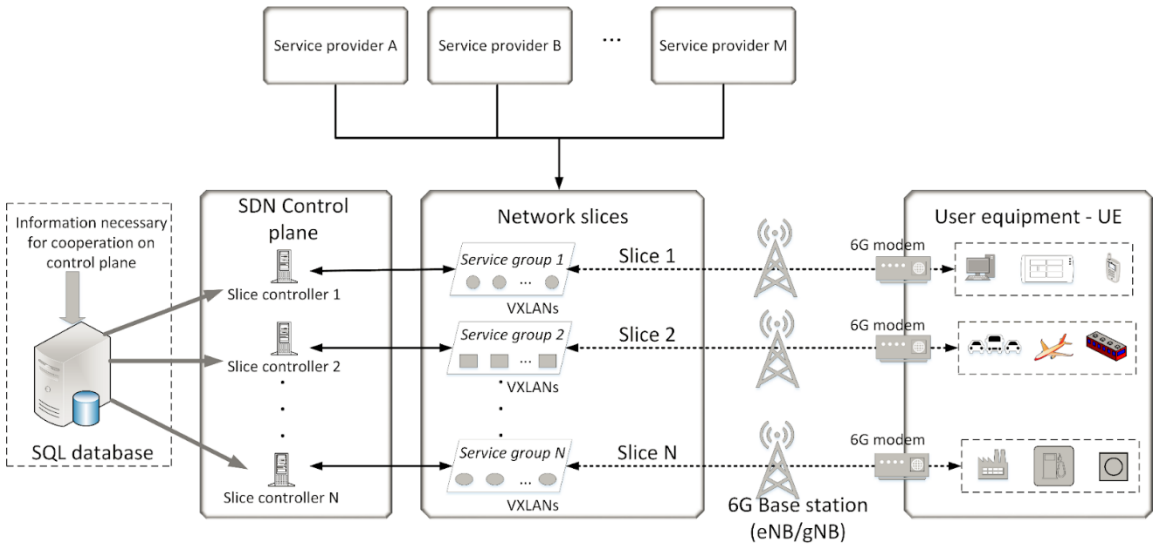
Ovakav pristup bi značajno poslužio operateru mreže i korisnicima, jer bi eliminisao nekoliko ograničavajućih faktora. Naime, operater proširuje i obogaćuje svoje usluge dozvoljavajući različitim provajderima da ponude istu grupu servisa. Na ovaj način korisnici takođe imaju koristi zbog mogućnosti izbora provajdera. Primena takvog rešenja unutar 5G/6G mreže implicitno podstiče konkurenciju i snažno postavlja korisničko iskustvo u prvi plan.

8.2 Arhitektura mreže i metodologija primene koncepta hibridnog SDN-a

Primena slajsova u mrežnoj arhitekturi trebalo bi da poboljša interoperabilnost mreže provajdera bežične infrastrukture i provajdera servisa kako bi se efikasno odgovorilo na zahteve korisnika, što je usko povezano sa obezbeđivanjem boljeg QoE kroz adekvatan QoS. Prethodno istraživanje uglavnom je koristilo postojeće redove čekanja ili druge rezervisane resurse u hardveru za prosleđivanje saobraćaja, što je u suprotnosti sa SDN paradigmom, čiji je primarni cilj povećanje programabilnosti mreže. Imajući to u vidu, mogu se izdvojiti sledeće činjenice:

- Nijedan SDN kontroler ne omogućava standardizovano upravljanje redovima (osim dodeljivanja tokova redovima);
- *OpenFlow* i OVSDB predstavljaju utvrđene standardne protokole.

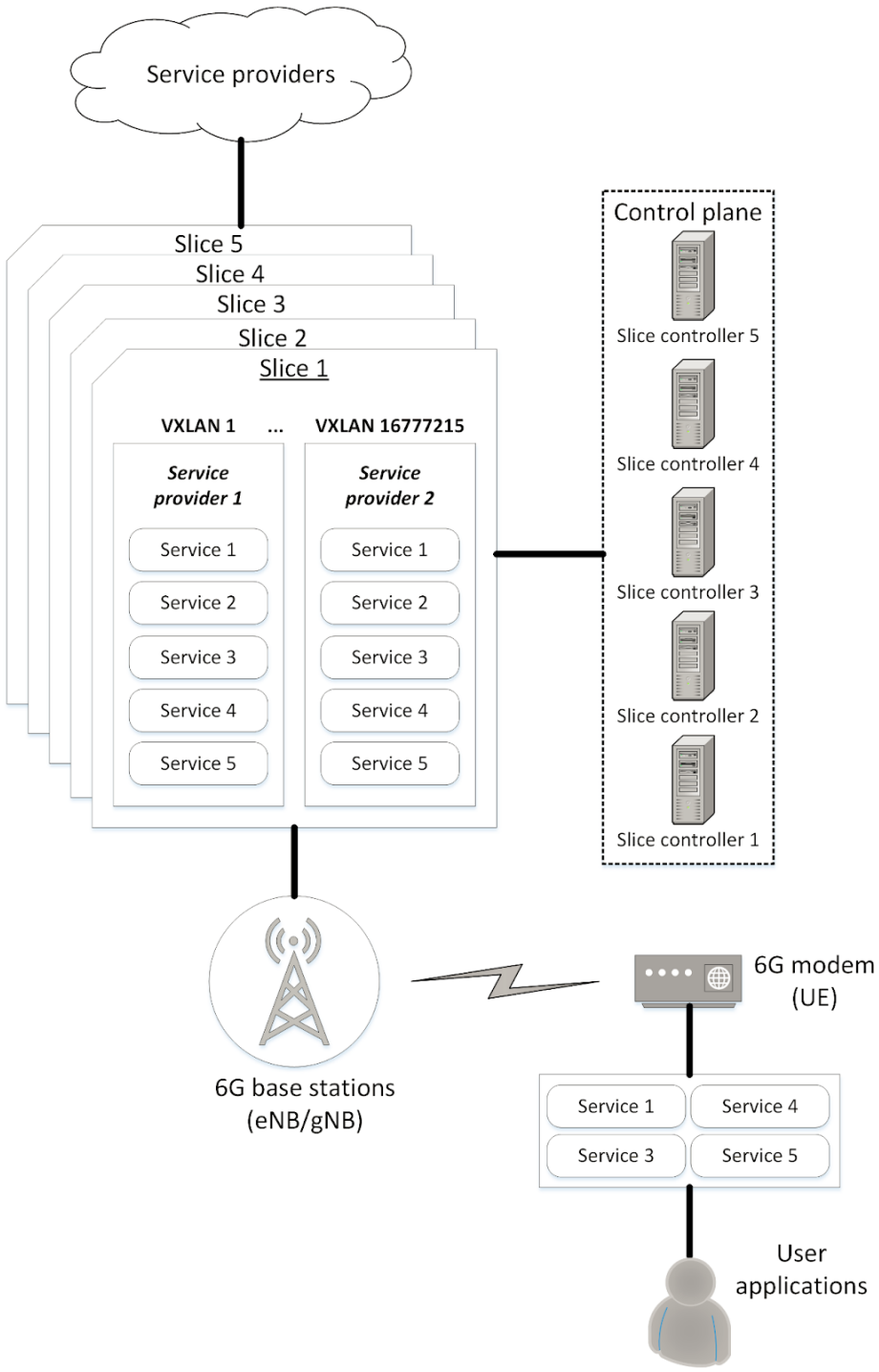
U bežičnom okruženju sa više slajsova, poseban akcenat se mora staviti na problem upravljanja saobraćajem i u tom smislu definisati odgovarajuća organizaciju slajsova u budućim 6G mrežama (slika 8.1). Prvo, moraju se povezati srodni servisi koje najverovatnije nudi isti provajder servisa. Razumno je izbegavati korišćenje nepotrebnih resursa za kreiranje posebnog slajsa za svaki servis. Usluge grupisanja mogu da obezbede efikasno deljenje resursa u realnom vremenu prema poslovnim procesima. Svaka grupa predstavlja poseban slajs koji kontroliše njen autoritativni kontroler. U cilju pojašnjavanja uloge i važnosti povezivanja servisa u *multi-slice* 5G/6G mreži, predstavljen je slučaj primene u podešavanju inteligentnih vozila.



Slika 8.1 Predloženi model 6G *multi-slice* okruženja

Iz dosadašnjeg iskustva se može pretpostaviti, da će različiti provajderi nuditi iste grupe servisa na tržištu. U okviru ovog istraživanja, predložen je koncept da se u okviru jednog slajsa „nađu" različiti provajderi servisa koji korisnicima pružaju servise ili grupe servisa sa istim ili sličnim servisnim zahtevima. Unutar svakog pojedinačnog slajsa, usluge i provajderi servisa se mogu izolovati primenom postojećih mehanizama segmentacije mreže (npr. VLAN, VXLAN). Ovo rešenje omogućava visok nivo skalabilnosti pošto broj slajsova nije ograničen. Predlaže se korišćenje VXLAN-a kako bi se tehnički odvojili provajderi koji nude istu grupu servisa (u svakom slajsu može postojati 224 različita provajdera koji pripadaju jednoj grupi servisa). Na taj način, korisnicima se pruža mogućnost, da pristupe različitim slajsovima i izaberu po njihovim kriterijumima najisplativije ponude. Drugim rečima, korisnici dobijaju mogućnost izbora provajdera servisa za svaku grupu servisa.

Višestruko kreiranje slajsova u mreži donosi sa sobom E2E koncept, implementiran u svim segmentima mreže (RAN, *edge, core* i *transport*) [108]. SDN funkcionalnost 5G/6G mreže je dostupna sve od *edge* mreže, pa do 6G modema na strani korisnika. Ovaj koncept infrastrukture omogućava potpuno otvoren pristup tržištu servisa, što ide u prilog i klijentima i provajderima servisa (slika 8.2).



Slika 8.2 Spajanje servisa i izolacija provajdera u *multi-slice* okruženju

Imajući u vidu da je neophodno uspostaviti sveobuhvatnu kontrolu dinamičkog kvaliteta servisa unutar svakog slajsa, podrazumeva se da u realizaciji tehničkog rešenja jedan SDN kontroler kontroliše jedan slajs. Primenom zajedničke baze podataka, ostvaruje se sinhronizacija kontrolera slajsova. Ova baza podataka predstavlja dobro organizovan repozitorijum koji sadrži informacije o mrežnim funkcijama, resursima, provajderima servisa, arhitekturi i procesima orkestracije. Ove informacije podržavaju zakazivanje sličnih servisa u grupi servisa određenog mrežnog slajsa.

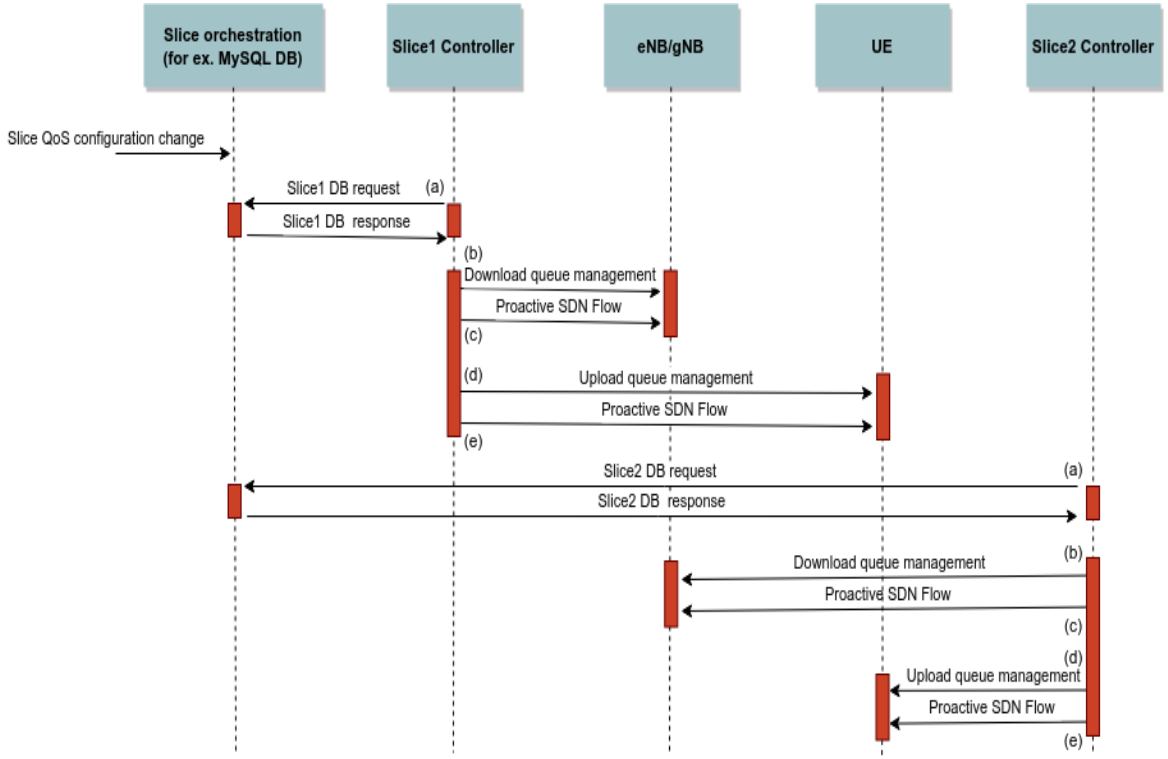
Predložena je metodologija dinamičkog upravljanja QoS-om u okviru koje se koristi dinamičko upravljanje redovima i primenjuju mehanizmi za donošenje inteligentnih odluka za maksimalne brzine servisnih grupa.

Rezervacija resursa obezbeđena je efektivnim kreiranjem redova za *upload* i *download*, čime se podmiruju zahtevi za propusnim opsegom za tu grupu servisa. Ako se zahtevi menjaju i korisnik otkaže pretplatu na određenu grupu servisa, rezervacija resursa za te servise treba da se poništi uklanjanjem redova za *upload* i *download*. Automatski, druge grupe servisa (slajsovi) mogu koristiti preostali oslobođeni propusni opseg. Usled mobilnosti korisnika očekuje se da će varijacije u kvalitetu i jačini signala između korisnika i baznih stanica prouzrokovati varijacije u trenutno dostupnom propusnom opsegu. Pored toga, trenutno zagušenje određene bazne stanice je faktor koji može dovesti do smanjenja propusnog opsega po korisniku. Stoga je neophodno sprovesti mehanizam koji dinamički upravlja QoS konfiguracijom slajsova osetljivih na gubitak paketa, i trebalo bi ih tretirati drugačije.

Uvođenjem dinamike u QoS konfiguraciju, može se primeniti sledeća strategija: neki manje kritični servisi (npr. internet servis) mogu biti privremeno usporeni ili čak suspendovani dok se kapacitet propusnog opsega veze ne poboljša. Važno je napomenuti, da u okviru istraživanja nije bio procenjen kapacitet propusnog opsega bežične veze, već je pretpostavljeno da se dostupna širina propusnog opsega može predvideti iz parametara prenosa (npr. jačina signala, odnos signal-šum).

Imajući u vidu sve što je prethodno navedeno, jasno se može izvući zaključak da se dinamički QoS može obezbediti kroz primenu metode dinamičkog upravljanja redovima. To podrazumeva mogućnost, da se dinamički menja konfiguracija parametara redova. Sve promene se vrše isključivo u skladu sa zahtevima specificiranim kroz orkestraciju slajsova na aplikacionom sloju. Kako bi se postigao ovaj cilj, predlaže se primena OVSDB protokola za proširenje funkcionalnosti SDN kontrolera, pored funkcija koje podržava *OpenFlow* protokol. Naime, primenom OVSDB protokola, vrši se dinamičko upravljanje redovima (npr. dinamičko kreiranje, brisanje i promena parametara reda).

Omogućava se dinamička kontrola propusnog opsega veze, prateći SDN paradigmu, kroz kombinaciju funkcionalnosti sadržanih u *OpenFlow*-u (dodeljivanje toka reda) i OVSDB protokolu. Kako bi QoS bio dvosmerno regulisan, redovi bi trebalo da budu postavljeni na oba SDN sviča: na strani bazne stanice (za *download*) i na strani korisnika (za *upload*). Predlog za dinamičko upravljanje QoS-om prikazan je na dijagramu toka na slici 8.3.



Slika 8.3 Dijagram toka promene QoS politike

Proces prikazan na slici 8.3 počinje zahtevom za promenu QoS politike za specifičnu grupu servisa u bazi podataka za orkestracije slajsova. Ovaj proces sadrži pet koraka koje svaki kontroler slajsova preuzima nezavisno od ostalih kontrolera:

a) Kontroler slajsa preuzima politiku QoS-a za slajs za koji je zadužen, i ako ima bilo kakvih promena u poređenju sa prethodnim stanjem, izvršava te promene;
b) Kontroler slajsa šalje baznoj stanici komandu za izmenu reda preko OVSDB protokola;
c) Kontroler slajsa stvara proaktivni tok za dodeljivanje saobraćaja slajsa odgovarajućem redu na baznoj stanici;
d) Kontroler slajsa šalje komandu za izmenu reda korisničkoj opremi (UE) preko OVSDB-a protokola;
e) Kontroler slajsa stvara proaktivni tok za dodeljivanje saobraćaja slajsa odgovarajućem redu na UE.

Dinamičko upravljanje redovima možda neće biti dovoljno u borbi sa budućim izazovima mreža sa visokim zahtevima. Za posmatranje, i zahteva servisa, i mogućnosti povezivanja, potreban je viši nivo inteligencije i to pre donošenja ispravnih odluka za maksimalne brzine i veličine redova. Stoga ovo istraživanje predlaže mehanizam za dodelu propusnog opsega veze koji koristi informacije

uskladištene u bazi podataka i izračunava maksimalne brzine za *download* i *upload* slajsova svakog korisnika. Za svakog korisnika *i* i slajs *j*, veličina reda (maks. brzina) računa se pomoću formule (12):

$$q_{ij} = \begin{cases} c_i w_{ij}, & t_{ij} \leq c_i w_{ij} \wedge s_{ij} + c_i w_{ij} \leq c_i \\ m_{ij}, & t_{ij} > c_i w_{ij} \wedge s_{ij} + m_{ij} \leq c_i \\ c_i - s_{ij}, & (t_{ij} \leq c_i w_{ij} \wedge s_{ij} + c_i w_{ij} > c_i) \\ & \vee (t_{ij} > c_i w_{ij} \wedge s_{ij} + m_{ij} > c_i) \end{cases} \tag{12}$$

$$m_{ij} = \min\{t_{ij},\ c_i - \sum_{k,l} t_{kl} \mid p_{kl} < p_{ij}\} \tag{13}$$

$$s_{ij} = \sum_{k,l} \max\{c_k w_{kl},\ t_{kl}\} \mid p_{kl} < p_{ij} \tag{14}$$

gde je t_{ij} minimalna brzina *download*/*upload*-a za korisnika *i* i slajs *j*; c_i je *download*/*upload* kapacitet veze za korisnika *i*; w_{ij} je udeo kapaciteta veze (procenat); p_{ij} je prioritet za slajs *j*, korisnika *i*; konačno, m_{ij} i s_{ij} se izračunavaju prema formulama (13) i (14).

Prema formuli (12), izračunava se veličina reda na osnovu tri kriterijuma:

1. Kapacitet veze je dovoljan da se zadovolje svi zahtevi jednog slajsa;
2. Proporcionalna alokacija propusnog opsega je manja od minimalne potrebne brzine u okviru slajsa (13);
3. Zahtev za slajsom nije mogao biti zadovoljen zbog alokacije smanjenih kapaciteta veze i propusnog opsega slajseva višeg prioriteta (14).

U slučajevima kada je proporcionalna alokacija manja od minimalne brzine, koristi se minimalna brzina. Međutim, ako su slajsovi višeg prioriteta i već im je dodeljen propusni opseg prema prva dva kriterijuma, slajs može da koristi samo preostalu neraspoređenu brzinu.

Izračunava se i dodela propusnog opsega veze u opadajućem redosledu prioriteta. Dakle, dodeljuje se propusni opseg za slajsove sa višim prioritetom pre onih sa nižim prioritetom.

8.3 Dizajn okruženja za testiranje

Sledeći korak jeste dizajniranje adekvatnog okruženja za testiranje prema predloženoj metodi i procena performansi prilagođenog kreiranja slajsova. Nakon toga se proverava prednost dinamičkog upravljanja QoS-om. Cilj je da se primeni predložena metoda, tako da dinamičko upravljanje bude

jednostavno, ograničeno na ono što je od interesa za istraživanje. Dakle, istraživanje se ne bavi različitim tehnikama zakazivanja redova, tehnikama za ublažavanje zagušenja niti različitim politikama redova. Ovo rešenje koristi *Open vSwitch* redove sa *Linux Hierarchy Token Bucket*-om (Linux-HTB) [134] i ograničenje maksimalne brzine redova. Sledeći odeljci opisuju dizajn okruženja za testiranje, kako je implementirana predložena metodologija, način generisanja saobraćaja i scenarije testiranja za verifikaciju metodologije.

Kako bi se testirale i verifikovale mogućnosti predloženog rešenja neophodno je implementirati odgovarajuće okruženje sa više slajsova koje bi adekvatno oslikavalo okruženje 6G mreža. Zbog toga je postavljeno virtuelno okruženje primenom *EVE-NG* platforme u cilju predstavljanja pojednostavljenog okruženja sa više slajsova za buduće mreže. Dizajn okruženja za testiranje sastoji se od tri dela:

1. Infrastruktura 6G provajdera sastoji se od uređaja *core* i distributivnog sloja mreže i uređaja za kontrolu ravni upravljanja (SDN kontroleri i SDN bazne stanice);
2. Korisnici (npr. pametna vozila) pristupaju 5G/6G infrastrukturi preko bazne stanice;
3. Veze sa nezavisnim provajderima servisa.

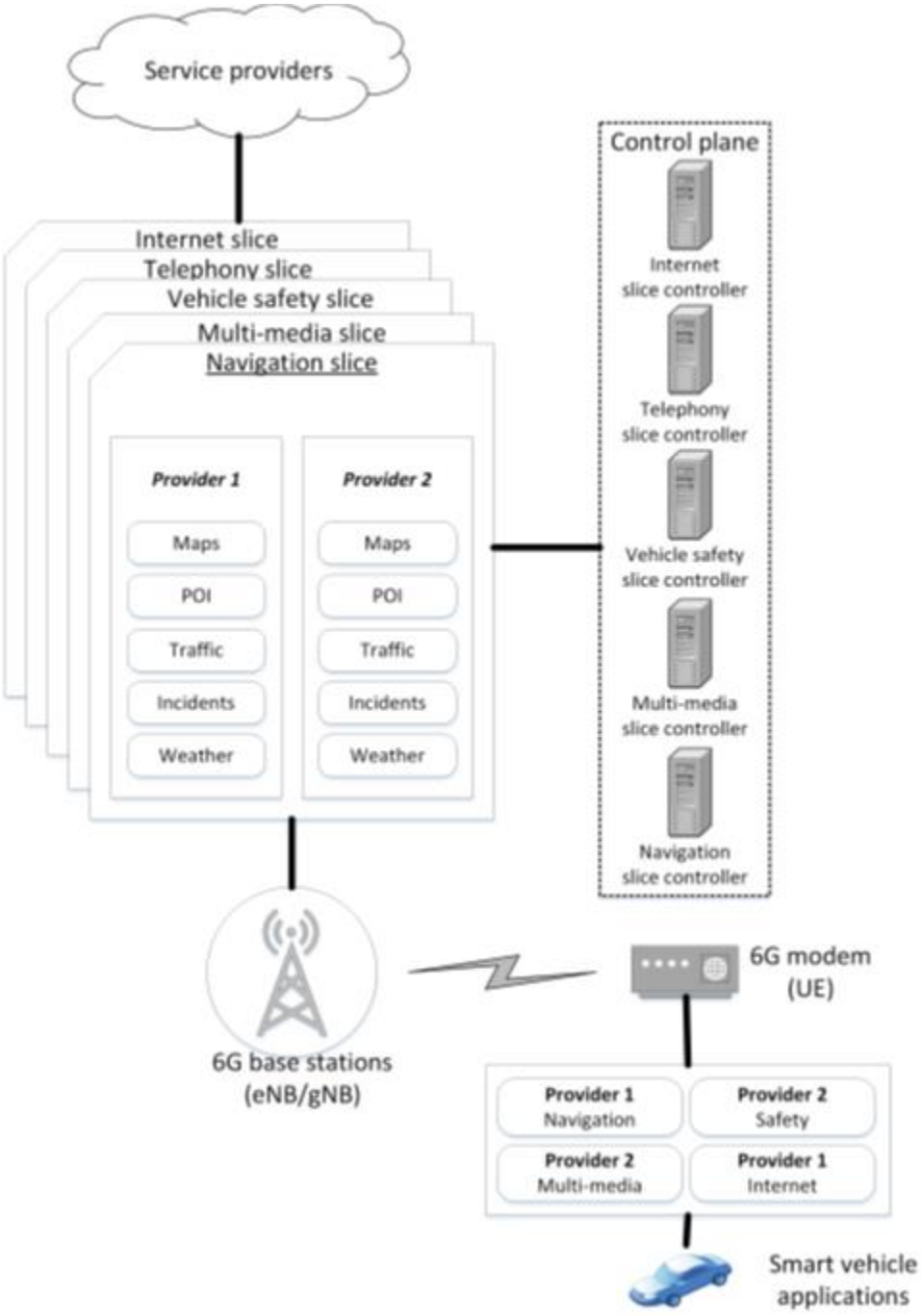
Predložena metodologija biće verifikovana primenom tri nezavisna slajsa. Radi jednostavnosti testne platforme, grupe servisa biće ograničene na jedan servis po slajsu. Svaki slajs je zadužen za jednu grupu servisa. Važno je naglasiti da poseban SDN kontroler upravlja svakim slajsom ili određenom grupom servisa. U cilju dobijanja realnijeg okruženja za testiranje, korisniku se pružaju usluge različitih provajdera. Tabela 8.1 predstavlja raspored slajsova sa servisima svake od grupa i zahteve za propusni opseg.

Naziv slajsa	**Grupa servisa**	**Download**	**Upload**
Internet	Web surfing	0.5-40 Mbps, 150-1500 pps	0-0.4 Mbps, 50-300 pps
Multimedia	Video streaming	5.5-6.2 Mbps, 1800-2200 pps	0-0.2 Mbps, 0-20 pps
Security	Sensors and surveillance stream	0-0.2 Mbps, 0-20 pps	2.8-3.2 Mbps, 820-940 pps

Tabela 8.1 Raspored slajsova u testnom okruženju

Za svaki slajs se implementiraju dve komponente na pametnom vozilu. Prva je *CarSDN* komponenta (SDN svič, integrisani deo 5G/6G modema) koja obezbeđuje programabilnost na strani korisnika, segmentaciju sa više slajsova i inteligentni red čekanja i upravljanje saobraćajem. Primena dinamičkog kreiranja redova i mogućnost promene njihovih parametara omogućavaju

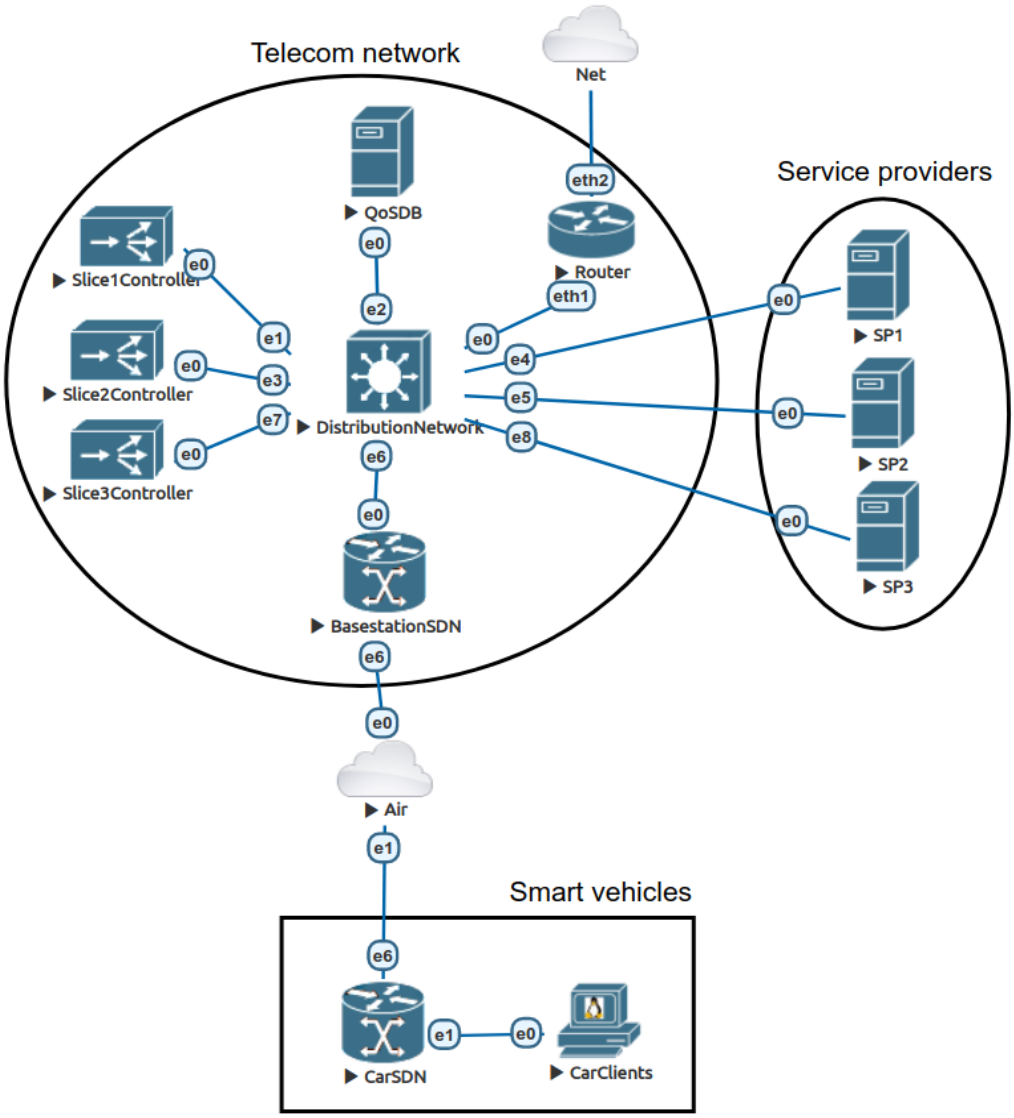
automatizovano upravljanje QoS mehanizmima. *Multi-slice* segmentacija je obezbeđena do *CarSDN* komponente (5G/6G modem). Primer takvog *multi-slice* okruženja dat je slikom 8.4.



Slika 8.4 Primer *multi-slice* okruženja u pametnom vozilu

Jedna od uloga ove komponente jeste da optimalno upravlja QoS-om odlaznih komunikacija i da omogući dinamičku alokaciju dostupnog *upload* propusnog opsega. Klijentske aplikacije su druga komponenta. One su ugrađene u ECU (elektronsku upravljačku jedinicu) automobila ili terminalnu opremu korisnika unutar automobila.

Na slici 8.5, prikazan je još jedan uređaj kojim se može upravljati između bazne stanice (*BasetationSDN*) i automobila (*CarSDN*). Ovaj uređaj ima ulogu uvođenja ograničenja dostupnosti propusnog opsega i, na taj način, omogućavanja zagušenja u virtuelnom sistemu (zagušenje predstavlja promenljivu sposobnost prenosa u etru).



Slika 8.5 Dizajn okruženja za testiranje

Okruženje za testiranje uključuje i poseban server za upravljanje servisima (QoSDB) sa ulogom da omogući dinamičke promene QoS parametara i obezbedi neophodnu sinhronizaciju između kontrolera slajsova. Ovaj server sadrži SQL bazu podataka koja precizno određuje konfiguracione parametre željenih QoS politika. SDN kontroler zadužen za određeni slajs preuzima QoS politiku iz SQL baze podataka i snabdeva svičeve u mreži odgovarajućim redovima.

Implementacija predložene metodologije počinje analizom QoS politika u SQL bazi podataka koja se koristi kao izvor informacija potrebnih za dinamičko konfigurisanje QoS-a. Svaki unos u bazi podataka predstavlja grupu servisa koje pruža jedan slajs i sadrži sledeće atribute (slika 8.6):

- Jedinstvena identifikacija korisnika (npr. IMSI i IMEI);
- Jedinstveni identifikator slajsa;
- IP adresa provajdera grupe servisa;
- Potreban kapacitet veze za *download* (procenat) na nivou slajsa;
- Potreban kapacitet veze za *upload* (procenat) na nivou slajsa;
- Potreban minimalan propusni opseg za *download* na nivou slajsa;
- Potreban minimalan propusni opseg za *upload* na nivou slajsa;
- Indeks prioriteta za postavljanje redosleda alokacije propusnog opsega.

uid	sliceId	ispIp	downloadPercent	uploadPercent	downloadMin	uploadMin	priority
12345	11111	192.168.110.254	40	20	1000	1200	1
12345	22222	192.168.120.254	40	10	4000	200	3
12345	33333	192.168.130.254	20	70	200	2000	2

Slika 8.6 Uzorak sadržaja SQL baze podataka

Pored toga, rešenje uključuje logički odvojenu tabelu SQL baze podataka koja čuva predviđen ukupni kapacitet veza za *download* i *upload* svakog korisnika. Ove uskladištene vrednosti se koriste u proračunu maksimalnog propusnog opsega za svaki slajs. Ovi atributi se koriste za upravljanje redovima na *BasestationSDN* i *CarSDN* svičevima. ID slajsa se koristi za pretragu drugih atributa iz SQL baze podataka. Atributi politike slajsova se zatim koriste za izračunavanje brzine za *download* (maksimalan dozvoljen propusni opseg za *download*, dodeljen redu), a ona se postavlja na *BasestationSDN* sviču, dok se brzina za *upload* (maksimalan dozvoljen propusni opseg za *upload*, dodeljen redu) postavlja na *CarSDN* sviču. IP adresa servera grupe servisa koristi se za definisanje proaktivnih tokova, na kojima će biti primenjeni odgovarajući redovi. Softversko rešenje za dinamički QoS napisano je u *Python*-u i dostupno je javnosti [135]. Ovo rešenje je zaduženo za konfiguraciju redova i tabele tokova na SDN svičevima prema unosima baze podataka. Sinhronizacija između željenog stanja (definisanog u SQL bazi podataka) i stvarnog stanja na svičevima (redovi i tabela tokova) se postiže periodičnim izvršavanjem *Python* skripte. Pokreće se po jedna instanca *Python* skripte na svakom od SDN kontrolera. Svaki SDN kontroler je odgovoran za svoj slajs, i takođe je odgovoran za oba SDN sviča kada su u pitanju servisi konkretnog slajsa. Stoga svrha *Python* skripte je da rukuje dinamičkim delom konfiguracije QoS-a na SDN svičevima. *Python* skripta prihvata dva ulazna parametra:

1. ID slajsa;
2. Period nakon kojeg se program skripte ponavlja (u sekundama) - ovo predstavlja periodu provere sadržaja baze podataka.

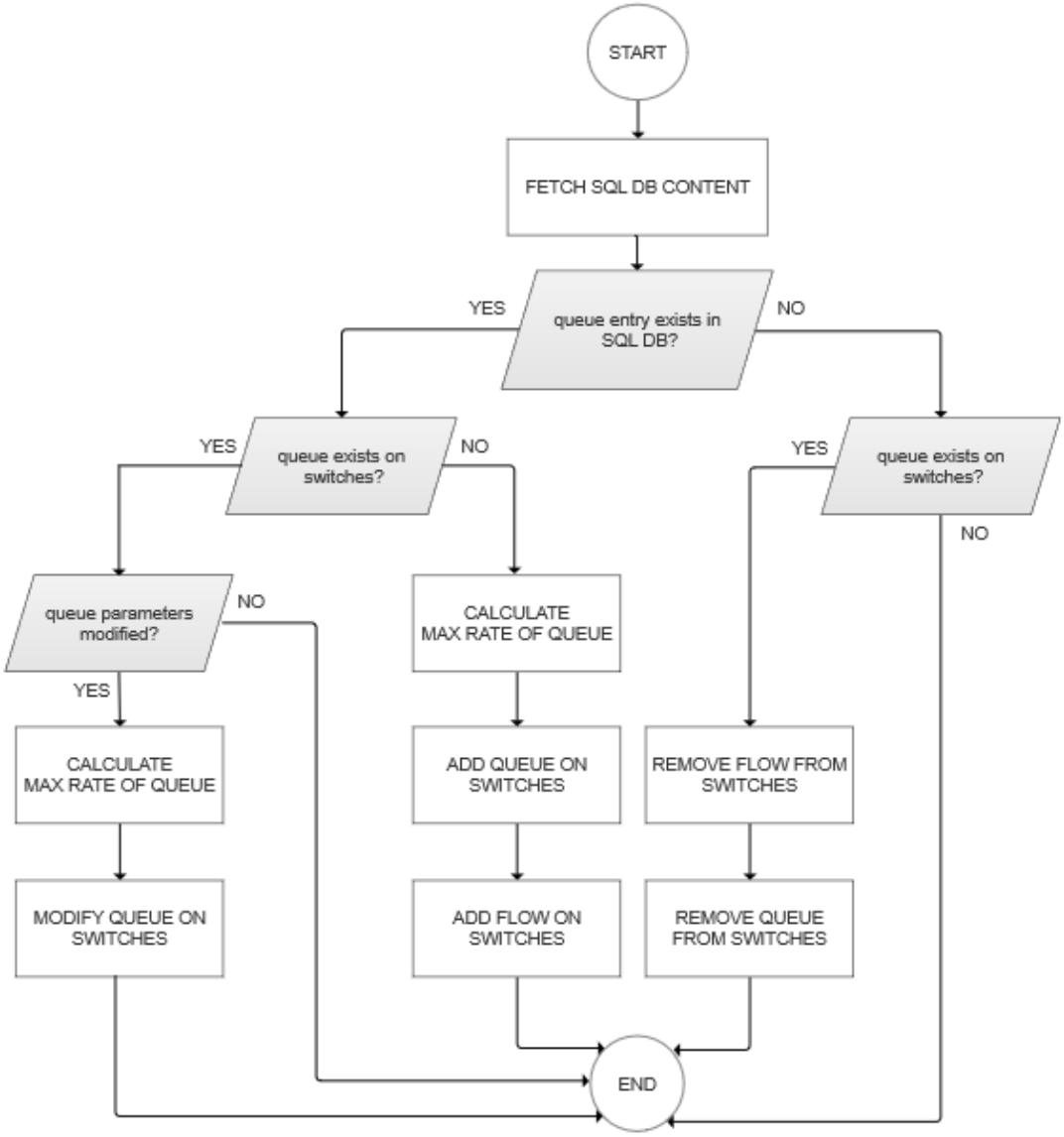
Redovi i proaktivni tokovi se konfigurišu pomoću komandne linije *ovs-vsctl* softvera (deo *Open vSwitch* alata). Implementacija predložene metodologije opisana je algoritmom 8.1, a dijagram algoritma prikazan je na slici 8.7.

Algorithm 1: Dynamic QoS Algorithm

Input: current queue query in SQL table

```
1: if queue entry does not exist in SQL table then
2:     if queue entry does not exist on switches then
3:         do nothing
4:     else
5:         delete proactive flows on switches
6:         delete queues on switches
7:     end if
8: else
9:     if queue entry does not exist on switches then
10:        calculate max-rate of queues
11:        add queues on switches
12:        add proactive flows on switches
13:    else
14:        if current query result ≠ previous query result then
15:            calculate max-rate of queues
16:            modify queues on switches
17:        end if
18:    end if
19: end if
```

Algoritam 8.1 Pseudokod algoritma dinamičkog QoS



Slika 8.7 Dijagram algoritma dinamičkog QoS

Prvi korak jeste preuzimanje sadržaja SQL baze podataka. Svaka promena *download* ili *upload* parametara će rezultirati konfiguracijom odgovarajućih redova na oba sviča. Ako redovi nisu konfigurisani za dati slajs, mora se kreirati novi red sa svojim parametrom maksimalne propusnosti za svaki od svičeva. Takođe se dodaju proaktivni tokovi koji podržavaju ove nove redove na svičevima. U nekim slučajevima, određeni unos reda ne postoji u SQL bazi podataka, ali njegova konfiguracija postoji na svičevima. Skripta u tom slučaju uklanja i redove i tokove sa svičeva. Kao što je već pomenuto, program se ponavlja, a drugi parametar skripte definiše period ponavljanja. Vremenski interval za osvežavanje politike QoS-a zavisi od politike implementacije dinamičkog QoS-a. Tako, na primer, QoS politike ne moraju da se često ažuriraju za pretplate i odjave na grupe servisa. S druge strane, kada je potrebno obezbediti brz odgovor na promenjeno stanje kvaliteta veze, neophodna je podrška brzog ažuriranja QoS parametara. Testna platforma ovog istraživanja

predstavlja slučaj kada je brza promena politike QoS-a nepotrebna, tako da se ažuriranje vrši svake sekunde.

Umesto simulacije, kreirani su emulacija okruženja i upravljanje saobraćajem, implementacijom odgovarajućeg softvera na serverima kako bi se obezbedilo adekvatno okruženje za testiranje. Radi bolje organizacije arhitekture servisa, grupe servisa su segmentirane kreiranjem različitih slajsova i raspoređivanjem saobraćaja na osnovu parametara iz tabele 8.2.

Slajs	**ID slajsa**	**Tip saobraćaja**	**Alat za evaluaciju**
Internet (NET)	11111	HTTP TCP	Apache/PHP script [135]
Multimedia (MM)	22222	UDP Stream	D-ITG [136]
Security (SEC)	33333	UDP Stream	D-ITG

Tabela 8.2 Parametri saobraćaja slajsova

Na serveru SP1 instaliran je *Apache web* servis i PHP engine za generisanje saobraćaja za internet (NET) slajs. Zatim je kreirana PHP skripta koja generiše odgovor nasumične veličine između 0,05 i 5 megabajta za svaki *web* zahtev [135] generisan jednom svake sekunde. Takav generator realno simulira internet saobraćaj i potrebe korisnika za surfovanje *web*-om i preuzimanje *web* resursa različitih veličina. Pošto korisnici uglavnom koriste HTTP protokol, *web* servisi se oslanjaju na TCP, što im omogućava da koriste ugrađene mehanizme za kontrolu tokova podataka, pa čak i da se dinamički prilagođavaju trenutnoj dostupnosti propusnog opsega transportne veze (kroz *windowing* mehanizam).

Implementirana su još dva slajsa (grupe servisa). Ova dva slajsa su nazvana multimedijalnim i bezbednosnim slajsovima. Multimedijalni (MM) slajs oslanja se na *multicast* saobraćaj ili *Video-on-Demand* (VoD) prenos video signala. Oni koriste UDP strimovanje i nemaju automatske mehanizme za kontrolu tokova. Za generisanje saobraćaja sa neujednačenom eksponencijalnom raspodelom primenjen je D-ITG alat [136]. U smeru od provajdera servisa ka korisniku generisano je oko 2000 paketa u sekundi sa ukupnom propusnošću između 5,5–6,2 Mbps.

Bezbednosni slajs (SEC) ima ulogu da vrši konstantno video strimovanje signala za nadzor i vrednosti senzora na *cloud* „crne kutije". Svrha ovih servisa jeste da omogući analizu u realnom vremenu i time omogući forenziku neželjenih događaja i optimizuje rad svih *engine*-a u vozilu. U smeru od vozila ka provajderu servisa, generisan je saobraćaj od oko 900 paketa u sekundi i ukupna propusnost od 2,8–3,2 Mbps. Slično MM slajsu, primenjen je D-ITG alat za generisanje eksponencijalne raspodele saobraćaja.

Kako bi se procenilo predloženo rešenje za dinamički QoS u mreži sa više slajsova, definisani su različiti scenariji testiranja i njihovi parametri (tabela 8.3):

- Scenario 1 - predstavlja osnovni slučaj testiranja sa dovoljnim kapacitetom propusnog opsega za dve aktivne grupe servisa. SQL baza podataka se sastoji od samo dve QoS politike koje slajs kontroleri koriste za kreiranje instrukcija za upravljanje određenim SDN komponentama;
- Scenario 2 - uvodi treći deo koji zahteva dinamičko kreiranje redova i preraspodelu ukupnog kapaciteta veze sa i dalje dovoljno propusnog opsega za zahteve svih slajsova;
- Scenario 3 - uvodi zagušenje u testnom okruženju ograničavanjem propusnog opsega veze koji dele grupe servisa; na taj način se pokreće preraspodela propusnog opsega i testira predložena metodologija;
- Scenario 4 - primenjuje različite QoS politike koje pokazuju dinamičku prirodu prilagođavanja politike u okruženju smanjenog kapaciteta;
- Scenario 5 - ima za cilj da pokaže ponašanje rešenja kada je potrebno obavljati dinamičko oslobađanje resursa servisne grupe. Oslobađanje resursa je neophodno ako servisna grupa više nije aktivna, a dinamička rekonfiguracija slobodnih resursa može se vršiti za aktivne grupe servisa.

	Scenario 1			**Scenario 2**			**Scenario 3**			**Scenario 4**			**Scenario 5**		
Slice	WEB	MM	SEC	WEB	MM	SEC	WEB	MM	SEC	WEB	MM	SEC	WEB	MM	SEC
Service groups running	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓
Download %	90	×	10	50	40	10	50	40	10	40	40	20	80	×	20
Upload %	50	×	50	30	20	50	30	20	50	20	10	70	30	×	70
Download Min (Kbps)	2000	×	500	2000	6000	500	2000	6000	500	1000	4000	200	1000	×	200
Upload Min (Kbps)	500	×	2000	500	500	2000	500	500	2000	1200	200	2000	1200	×	2000
Priority	3	×	1	3	2	1	3	2	1	1	3	2	1	×	2
Link bandwidth cap	100/100 Mbps			100/100 Mbps			10/5 Mbps			10/5 Mbps			10/5 Mbps		

Tabela 8.3 Parametri različitih scenarija testiranja

Proces testiranja je automatizovan tako da je svaki scenario testiranja aktivan 60 s. Nakon toga, prelazi se na sledeći scenario.

Snimanje performansi postavljenog okruženja za testiranje obavljeno je pomoću sledećih alata:

- *bwm-ng* alat—koristi se za praćenje i evidentiranje aktivne propusnosti fizičkih interfejsa [137] na strani *CarClient*-a (za potrebe praćenja *download*-a) i *DistributionNetwork* sviča (za potrebe praćenja *upload*-a);
- D-ITG alat [136]—koristi se za evidentiranje i analizu QoS parametara (kašnjenje, džiter i gubitak paketa) UDP strimovanja na strani prijemnika na *CarClient*-u i SP3 serveru;
- *tcpdump* alat—koristi se za prikupljanje kompletnih evidencija saobraćaja u procesu testiranja na *DistributionNetwork* sviču.

Za SDN kontroler svakog slajsa, pokreće se po jedna instanca *Python* skripte za dinamičko upravljanje QoS-om, koja nadgleda SQL tabelu za samo jednu grupu servisa (ID slajsa). Ako postoji više provajdera istih servisa, potrebno je pokrenuti više instanci sa različitim ID-jem slajsa. U ovdašnjem slučaju testiranja, ove instance su pokrenute sa intervalom povlačenja podataka iz SQL baze od jedne sekunde, kako bi se postigao brži odgovor.

8.4 Rezultati testiranja i njihova analiza

Proces testiranja ima za cilj da proceni korišćenje resursa predložene metode dinamičkog upravljanja QoS-om u realnom okruženju. Kao dokaz koncepta izvršeno je više ciklusa za verifikaciju ispravnog rada svakog aspekta predloženog rešenja.

Primenjujući politike QoS baze podataka na predložene scenarije testiranja, dobijena je tabela 8.4. Za implementirano rešenje izračunate su maksimalne brzine preuzimanja i otpremanja podataka (*download*-a i *upload*-a) za svaki slajs i scenario, prateći predloženu proceduru alokacije. Većina vrednosti maksimalne brzine je dodeljena primenom prvog kriterijuma u (1). U scenarijima 3 i 4, gde je kapacitet veze smanjen ispod kriterijuma zahteva slajsa, može se posmatrati upotreba drugog i trećeg kriterijuma alokacije iz (1) (podebljano i podvučeno u tabeli 8.4, respektivno). Izračunate maksimalne brzine se zatim dinamički primenjuju na odgovarajući SDN svič pri modifikaciji reda.

	Scenario 1			Scenario 2			Scenario 3			Scenario 4			Scenario 5		
Slice	WEB	MM	SEC	WEB	MM	SEC	WEB	MM	SEC	WEB	MM	SEC	WEB	MM	SEC
Download mpbs	90	/	10	50	40	10	3	**6**	1	4	4	2	8	/	2
Upload mpbs	50	/	50	30	20	50	1.5	1	2.5	**1.2**	0.3	3.5	1.5	/	3.5

Tabela 8.4 Izračunate maksimalne brzine pri različitim scenarijima testiranja

Slika 8.8 prikazuje parametre reda čekanja bazne stanice u scenariju 2. Takođe prikazuje maksimalnu veličinu reda (maks. brzinu), minimalnu veličinu reda (min. brzinu), brzinu *burst*-a i statistiku za prenesene pakete, bajtove i greške za svaki slajs.

```
root@ubuntu:~# ovs-appctl -t ovs-vswitchd qos/show ens9
QoS: ens9 linux-htb
max-rate: 10000000000

Default:
  burst: 12512
  min-rate: 12000
  max-rate: 10000000000
  tx_packets: 1
  tx_bytes: 42
  tx_errors: 0

Queue 1:
  burst: 12512
  min-rate: 12000
  max-rate: 50000000
  tx_packets: 139814
  tx_bytes: 656259521
  tx_errors: 0

Queue 2:
  burst: 12512
  min-rate: 12000
  max-rate: 40000000
  tx_packets: 0
  tx_bytes: 0
  tx_errors: 0

Queue 3:
  burst: 12512
  min-rate: 12000
  max-rate: 10000000
  tx_packets: 7
  tx_bytes: 465
  tx_errors: 0 _
```

Slika 8.8 Vrednosti parametara redova na OVS baznoj stanici tokom scenarija 2

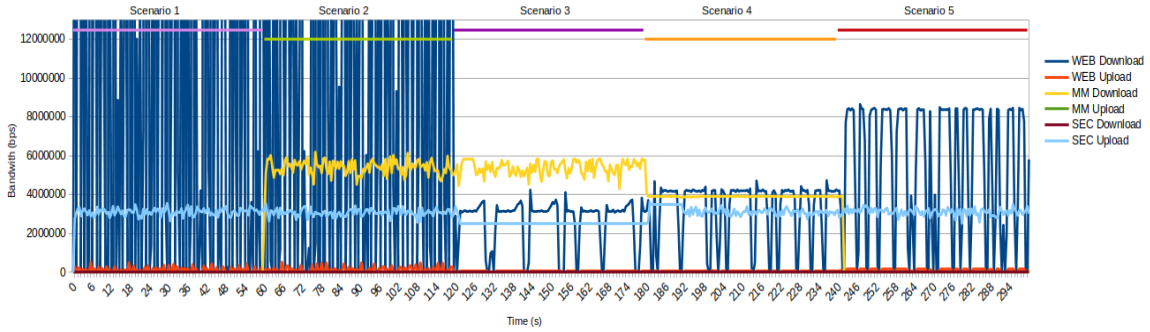
Dalje, na slici 8.9, prikazan je sadržaj tabele tokova, sa proaktivnim tokovima na OVS baznoj stanici u scenariju 2.

```
root@ubuntu:~# ovs-ofctl dump-flows br-sdn
 cookie=0x0, duration=3051.506s, table=0, n_packets=241234, n_bytes=637894707, ip,nw_src=192.168.110.254 actions=set_queue:1,NORMAL
 cookie=0x0, duration=2257.723s, table=0, n_packets=7, n_bytes=493, ip,nw_src=192.168.130.254 actions=set_queue:3,NORMAL
 cookie=0x0, duration=205.195s, table=0, n_packets=0, n_bytes=0, ip,nw_src=192.168.120.254 actions=set_queue:2,NORMAL
 cookie=0x0, duration=1131790.944s, table=0, n_packets=1307231, n_bytes=725654400, priority=0 actions=NORMAL
```

Slika 8.9 Tabela tokova na OVS baznoj stanici tokom scenarija 2

Tokom ovog procesa izvršava se više ciklusa. Primećeni su dosledni rezultati u pogledu izmerenih parametara performansi, a jedino uočeno odstupanje uzrokovano je nasumičnom veličinom odgovora WEB servisa.

Na slici 8.10 prikazani su rezultati testova koji pokazuju iskorišćenost propusnog opsega u različitim scenarijima testiranja.



Slika 8.10 Korišćenje propusnog opsega tokom procesa testiranja

Prvi scenario predstavlja osnovno stanje okruženja gde sve grupe servisa poseduju dovoljan protok. WEB servis zauzima do 40 Mbps, tako da se svaka transakcija izvršava za manje od jedne sekunde. SEC servis za otpremanje strimovanja takođe demonstrira svoje ponašanje pod ovim osnovnim uslovima.

Drugi scenario uvodi novu grupu servisa (MM servis) definisanjem novog SQL zapisa. Nakon početka ovog scenarija, SDN kontroler, zadužen za posmatranje slajsa, inicira kreiranje novih redova za tu servisnu grupu. Novi MM servis ne trpi od degradacije performansi jer još uvek postoji dovoljno zajedničkog propusnog opsega za sav saobraćaj.

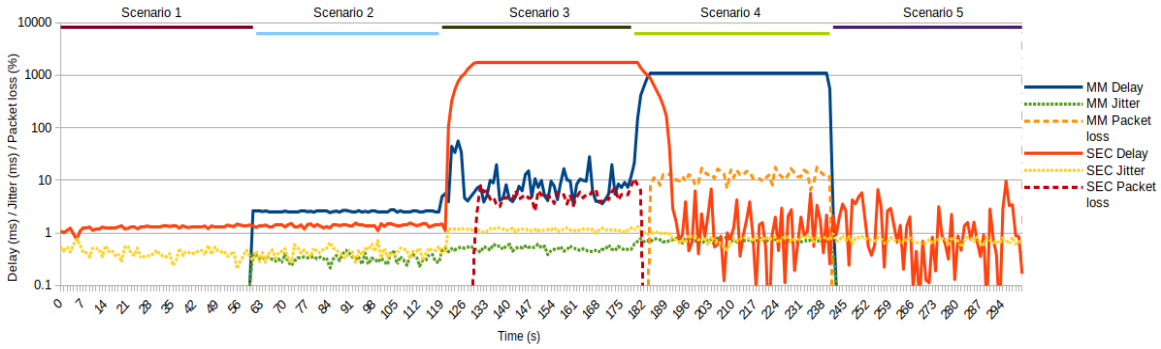
Treći scenario uvodi smanjenje propusnog opsega na deljenoj vezi, što uzrokuje zagušenje u oba smera (*upload* i *download* podataka). U pravcu *download*-a, veza ima maksimalni propusni opseg od 10 Mbps, dok WEB i MM usluge za neprekidan rad zajedno zahtevaju oko 47 Mbps propusnog opsega. Problem zagušenja može se odmah uočiti analizom odgovora na WEB zahteve jer se odgovori velikih veličina ne mogu završiti za manje od jedne sekunde. Umesto toga, prenos paketa mora biti preraspodeljen tokom vremena. Dostupni propusni opseg je preraspodeljen prema politikama u SQL

tabeli i predloženom mehanizmu alokacije propusnog opsega. Izdvaja se propusni opseg od 1 Mbps za SEC servis, 6 Mbps za MM servis i 3 Mbps za WEB servis. SEC i MM servisi imaju veći prioritet od WEB servisa. Dakle, WEB servis prima preostali (dostupni) propusni opseg (manji od onoga što zahteva propisana politika). Prema politici, jedina vidljiva promena u smeru *upload*-a jeste smanjenje propusnog opsega SEC servisa.

Rezultati dobijeni testiranjem u četvrtom scenariju ukazuju na efekte dinamičke promene u politici kvaliteta servisa. Dinamična promena SQL politike dovela je do promene prioriteta servisa, a WEB servis dobija najveći prioritet u alokaciji resursa. U pravcu *download*-a, WEB i MM servisima dodeljuje se 4 Mbps, a SEC servisu se dodeljuje 2 Mbps, u skladu sa QoS politikama. Međutim, dodela propusnog opsega je obavljena prema prioritetima servisa u pravcu *upload*-a. Ukupno je dodeljeno 1,2 Mbps za WEB servis, koji ima najveći prioritet, dok je za SEC servis predviđeno 3,5 Mbps. Preostali propusni opseg se dodeljuje MM servisu (manje od vrednosti definisane u politici QoS-a).

Peti scenario pokazuje situaciju u kojoj jedan od servisa više nije u upotrebi, pa je moguće izvršiti dinamičku preraspodelu oslobođenih resursa, kako bi se aktivnim grupama servisa dodelila veća propusnost i na taj način poboljšale performanse servisa. U ovom scenariju, propusni opseg koji je ranije dodeljen QoS politikom za MM servisa, sada se može preraspodeliti na preostale servise, što odmah daje povećan broj transakcija. Ovaj scenario potvrđuje da je moguće izbrisati redove grupa servisa koji više nisu potrebni.

Smanjenje propusnog opsega ispod nivoa koji zahtevaju servisi dovodi do degradacije performansi tih servisa. Slika 8.11 pokazuje uticaj zagušenja i primene QoS politika na performanse pojedinačnih grupa servisa.



Slika 8.11 Parametri QoS-a tokom procesa testiranja

Kao što se može primetiti, slika 8.10 jasno pokazuje da ograničenje *upload*-a SEC servisne grupe u trećem scenariju unosi značajno kašnjenje i gubitke paketa. Grupa servisa zahteva do 3,2 Mbps (tabela 8.1), ali QoS politika dodeljuje 2,5 Mbps. Stoga, kada su redovi popunjeni, svi budući

paketi iz ove grupe servisa se ispuštaju. Sličan slučaj je sa MM servisom, gde se saobraćaj generiše sa eksponencijalnom raspodelom do 6,2 Mbps, ali za *download* je rezervisano samo 6 Mbps.

Četvrti scenario uvodi još veće smanjenje propusnog opsega MM servisne grupe (4 Mbps za *download*), što rezultira značajnim kašnjenjem i gubitkom paketa. Međutim, ovo je pokazatelj loših performansi servisa za ovu uslugu strimovanja. Servisi aplikacija koji koriste UDP strimovanje često imaju neke mehanizme za dinamičko smanjenje protoka prenosa. Na primer, usluge video strimovanja koriste mogućnost dinamičke brzine prenosa. Kada klijent primeti degradaciju veze i gubitak paketa, može tražiti od servera da smanji brzinu prenosa (tj. smanji kvalitet video strima dok se gubitak paketa ne eliminiše). Scenario testiranja ovog istraživanja nije uključio takav koncept smanjenja brzine prenosa. Primećuje se da servisna grupa SEC u četvrtom i petom scenariju nije povratila zagušenje uvedeno u trećem scenariju.

9. Zaključak

Pod uticajem naglog porasta broja korisnika na internetu i neprestano rastuće složenosti njihovih zahteva, mrežne tehnologije se danas relativno brzo razvijaju. Implementacija modernih servisa zasnovanih na mašinskom učenju, IoT-u, neuronskim mrežama, i ostalim naprednim tehnologijama, kao i realizacija pametnih okruženja (gradova, saobraćaja, industrije) zahtevaju brzu i efikasnu modernizaciju mrežne infrastrukture. U mrežu je neophodno uključiti viši nivo programabilnosti i jednostavnije upravljanje njenom raznolikom infrastrukturom. Na ovaj način moguće je postići potreban nivo fleksibilnosti i proširivosti pri implementaciji postojećih, ali i budućih servisa.

SDN tehnologija nameće se kao logično rešenje po ovom pitanju. Međutim, neophodno je prevazići složenost postupka migracije na SDN tehnologiju, naročito po pitanju finansijskih izazova. Zato je potrebno na dovoljno jasan i precizan način definisati proces tranzicije sa tradicionalne na SDN mrežnu arhitekturu. Jedino trenutno racionalno rešenje sa stanovišta ekonomičnosti i tehničke efikasnosti jeste postepeno uvođenje SDN funkcionalnosti u mrežnu infrastrukturu. Ovo rešenje je zasnovano na selektivnoj implementaciji SDN mrežnih uređaja i izgradnji hibridne SDN mreže. Ovo istraživanje je predstavilo realizaciju jedne takve mreže dodavanjem samo jednog SDN sviča (sa tradicionalnim i SDN funkcionalnostima) i jednog SDN kontrolera, a istovremeno zadržavanjem svih tradicionalnih funkcionalnosti mreže (rutiranje i preusmeravanje paketa, DHCP, ARP, SNMP itd.).

U ovom istraživanju je realizovana i evaluirana šema za balansiranja opterećenja na *web* serverima u okviru hibridne SDN mreže zasnovane na SNMP protokolu. Ovim protokolom se nadgledaju trenutna opterećenja resursa istih *web* servera. U procesu odlučivanja kojem će *web* serveru pripasti opsluživanje datog klijenta, primenjena je metrika koja može uključiti više parametara opterećenja resursa. Ona je u stanju prihvatiti daleko veći broj parametara od klasičnih metoda za balansiranje opterećenja, pa čak i pojedinih novih SDN šema. Analizom dobijenih rezultata

testiranja, zaključeno je da je metoda implementirana u ovom istraživanju postigla bolje balansiranje opterećenja servera i obezbedila efikasniju implementaciju *web* servisa. Iako je predloženo rešenje bilo implementirano na *web* serverima, ono je takođe primenjivo, pa i efikasnije za balansiranje opterećenja i u drugim okolnostima.

Preostali deo istraživanja bavio se kreiranjem slajsova u mreži, što ima suštinsku ulogu u istraživanju koje ima za cilj razvoj novih koncepata fleksibilne mrežne arhitekture. Sadašnji nivo zrelosti i stalni razvoj i unapređenje ove tehnologije ukazuju da će *network slicing* uskoro postati mrežni standard i igrati vitalnu ulogu u budućem dizajnu i radu mreže. Dakle, razvoj efikasnijeg koncepta organizacije mreže u više slajsova od ključnog je značaja. Ovo istraživanje predlaže efikasan model koji omogućava prilagođeno kreiranje slajsova u budućim mrežnim arhitekturama. Osnova ovog modela je arhitektura na nivou slajsa i dinamičko upravljanje QoS-om.

Predloženo je rešenje koje podrazumeva povezivanje sličnih servisa u grupu i upravljanje njima kao jednim slajsom. Provajderi istih usluga spajaju se u isti slajs, na ovaj način mrežni slajs se može specijalizovati za specifične svrhe, pružajući tako određeni nivo fleksibilnosti arhitekture *core*-a mreže i ukazujući na to da potencijal ove arhitekture ide u prilog i mrežnom operateru i njegovom korisniku. Tako dobro organizovani i upravljani slajsovi podržavaju više provajdera servisa izolovanih u zasebne slajseve koristeći neke od dostupnih tehnologija segmentacije (npr. VXLAN, VLAN).

Evidentno je da aktuelna istraživanja prvenstveno koriste unapred definisani red čekanja i nedostaje im potpuno dinamičko upravljanje redovima. Takav pristup je suprotan SDN paradigmi, koja podrazumeva uvođenje sve više programiranja u mrežu. Ovo predstavlja ograničavajući parametar za fleksibilnost budućeg *core*-a mreže. Imajući to u vidu, ovim istraživanjem je predloženo novo rešenje koje eliminiše to ograničenje i primenjuje *high-level* programiranje pri upravljanju QoS-om. Svaki slajs podrazumeva infrastrukturne uređaje i poseban SDN kontroler koji kontroliše ovaj slajs. Primena veoma fleksibilnih softverskih rešenja može značajno smanjiti potrebu za poznavanjem fizičke mrežne infrastrukture. Takvo rešenje može omogućiti prilagođavanje ruta u mrežnim slajsovima na osnovu adekvatne QoS politike, predstavljajući dodatni stepen fleksibilnosti u 5G/6G infrastrukturi. Na ovaj način se u potpunosti pomera upravljanje QoS politikama na sloj aplikacije. Predstavljena metodologija uključuje mehanizam dodele propusnog opsega primenom složenih QoS politika (višeparametarska politika). Potrebno je naglasiti da u realnom okruženju, dinamikom prebacivanja između scenarija bi se upravljalo mehanizmima donošenja odluka zasnovanih na trenutnom saobraćaju, koji su van okvira testiranja ovog istraživanja.

Ovo istraživanje ne zalazi u to kako bi QoS politike trebalo da izgledaju u budućim mrežama jer ovo je i dalje veoma složena tema. Predloženi su mehanizmi za dinamičku implementaciju bilo koje planirane politike QoS-a, dozvoljavajući fleksibilne i dinamičke mrežne parametre zasnovane

na zahtevima SQL DB servisa. Ovo istraživanje se trudilo da zadovolji sve te zahteve predloženim mehanizmom.

Potrebno je skrenuti pažnju na određena ograničenja ovog naučnog istraživanja i ona su prouzrokovana načinom na koji se obavljalo testiranje predloženog rešenja. Testni scenario je realizovan u laboratorijskim uslovima gde je testno okruženje sačinjeno od minimalnog broja uređaja, odnosno mašina potrebnih za ostvarivanje ciljeva istraživanja. Prema tome, kada bi se testiranje obavljalo u realističnom ili *cloud* okruženju, rezultati bi mogli biti drugačiji.

Najznačajniji doprinos doktorske disertacije jeste definisanje metodologije, kojom se na efikasan, ali i ekonomski pragmatičan način, implementirala SDN funkcionalnost u tradicionalnim računarskim mrežama. Precizno su definisani principi i model inkrementalne primene SDN tehnologije u tradicionalnim računarskim mrežama i realizacija fleksibilne i skalabilne, hibridne SDN mrežne infrastrukture. U tako dizajniranoj infrastrukturi, moguće je, da se primenom naprednijih tehnika, znatno efikasnije balansira opterećenje na serverima (u odnosu na tradicionalne metode balansiranja). Metodologija izložena ovim istraživanjem omogućuje implementaciju i organizaciju veoma elastičnog okruženja, u kome se lako može implementirati nova šema balansiranja opterećenja. Ova šema balansiranja opterećenja zasnovana je na višeparametarskoj metrici i nudi bolje rezultate, u odnosu na one koji se dobijaju primenom tradicionalnih metoda ili LBBSRT modela balansiranja opterećenja. Ključni rezultati, odnosno naučni doprinosi jesu:

- Identifikacija i sistematizacija postojećih metoda primene SDN funkcionalnosti u cilju realizacije hibridne mrežne infrastrukture;
- Razvoj modela hibridne SDN mreže koji nadilazi po performansama tradicionalne mreže, a sa druge strane smanjuje kompleksnost rešenja same implementacije mreže;
- Dokaz o mogućnosti implementacije koncepta inkrementalne primene SDN funkcionalnosti u tradicionalnoj mrežnoj infrastrukturi, na osnovu rezultata dobijenih istraživanjima sprovedenim na *EVE-NG* platformi;
- Uporedna analiza rezultata dobijenih implementacijom postojećih algoritama za balansiranje opterećenja sa rezultatima dobijenim implementacijom algoritma zasnovanog na višeparametarskoj metrici u hibridnoj SDN mreži;
- Prezentacija statističkih podataka koji potvrđuju efikasnije balansiranje opterećenja primenom višeparametarske metrike u odnosu na postojeće metode balansiranja opterećenja.

Dalja moguća istraživanja bi se mogla fokusirati na razvoj dodatnih funkcionalnosti, koje bi poboljšale tradicionalne mreže računarskih sistema i rešavale probleme poput efikasnijeg pomeranja IP uređaja iz jedne u drugu mrežu. U skladu sa ovim problemom, istraživanje bi obratilo posebnu pažnju na razvoj modela raspodele težinskih koeficijenata (faktora), pogotovo u slučaju njihovog

uticaja na ogroman broj servera. Takođe, jedan deo istraživanja bi se osvrnuo i na precizniju identifikaciju uticaja predložene šeme za balansiranje opterećenja na saobraćaj sa malim kašnjenjem.

10. Literatura

[1] Sezer, S., Scott-Hayward, S., Chouhan, P., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., Rao, N.: *Are we ready for SDN? Implementation challenges for software-defined networks*. IEEE Commun. Mag. 51 (7) (2013) 36–43. http://dx.doi.otg/10.1109/mcom.2013.6553676

[2] Zhang, Z., Bockelman, B., Carder, D.W., Tannenbaum, T.: Lark: *An effective approach for software-defined networking in high throughput computing clusters*. Future Gener. Comput. Syst. 72 (2016) 105-117. http://dx.doi.org/10.1016/j.future.2016.03.010

[3] Bojović, Ž., Bojović, P., Šuh, J.: *Implementing Software Defined Networking in Enterprise Networks*. J. Inst. Telecommun. Prof. 12 (1) (2018) 30-35. https://doi.org/10.13140/RG.2.2.10305.86887

[4] Kaur, S., Kumar, K., Singh, J., Ghumman, N. S.: *Round-robin based load balancing in Software Defined Networking*, in: 2nd International Conference on Computing for Sustainable Global Development, INDIACom 2015, pp. 2136-2139.

[5] Cardellini, V., Colajanni, M., Yu, P. S.: *Dynamic load balancing on Web-server systems*. IEEE Internet Comput. 3 (3) (1999) 28-39. http://dx.doi.org/10.1109/4236.769420

[6] Zhong, H., Fang, Y., Cui, J.: LBBSRT: *An efficient SDN load balancing scheme based on server response time*. Future Gener. Comput. Syst. 68 (2017) 183-190. https://doi.org/10.1016/j.future.2016.10.001

[7] Göransson, P., Black, C., Culver, T., Chapter 5 - *The OpenFlow Specification*, Editor(s): Göransson, P., Black, C., Culver, T., Software Defined Networks (Second Edition), Morgan Kaufmann, 2017, pp. 89-136, ISBN 9780128045558, https://doi.org/10.1016/B978-0-12-804555-8.00005-3.

[8] Bojovic, P.D., Bojovic, Z., Bajic, D., and Vojin Šenk: *IP Session Continuity in Heterogeneous Mobile Networks Using Software Defined Networking*, 19(6) (2017) 563-568, https://doi.org/10.1109/JCN.2017.000096

[9] Case, J., Fedor, M., Schoffstall, M., Davin, J.: *A Simple Network Management Protocol (SNMP)*, Requset for Comments: 1157, 1990

[10] Information Sciences Institute: *Transmission Control Protocol, Darpa Internet Program Protocol Specification*, Request for Comments: 793, 1981

[11] Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., Uhlig, S.: *Software-Defined Networking: A Comprehensive Survey*. Proc. IEEE. 103 (1) (2015) 14 – 76. http://dx.doi.org/10.1109/JPROC.2014.2371999

[12] Singh, S., and Jha, R. K.: *A survey on software defined networking: Architecture for next generation network*. Journal of Network and Systems Management, 25(2)(2) 321-374. https://doi.org/10.1007/s10922-016-9393-9

[13] *Global OID reference database*, https://oidref.com Accessed 24 September 2022

[14] Case, J., McCloghrie, K., Rose, M., Waldbusser, S.:*Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*, Requset for Comments: 3416, 2002

[15] Harrington, D., Presuhn, R. B., Wijnen: *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks*, Request for Comments: 3411, 2002

[16] Peterson, L., Cascone, C., O'Connor, B., Vachuska, T., Davie, B.: *Software-Defined Networks: A Systems Approach*. Systems Approach LLC (2020)

[17] Scott-Hayward, S.: *Design and deployment of secure, robust, and resilient SDN controllers*, in: 1st IEEE Conference on Network Softwarization, NetSoft 2015, 2015. pp. 1-5. http://dx.doi.org/10.1109/NETSOFT.2015.7258233

[18] Kirkpatrick , K.: *Software-Defined Networking*, Communications of the ACM, vol. 56, no. 9 (2013) 16-19

[19] Lara, A., Kolasani, A., Ramamurthy, B.: *Network Innovation using OpenFlow: A Survey*. IEEE Communications Surveys & Tutorials. 16 (1) (2014) 493-512. http://dx.doi.org/10.1109/SURV.2013.081313.00105

[20] Yin, H., Zou, T., Xie, H.: *Defining Data Flow Paths in Software-Defined Networks with Application-Layer Traffic Optimization*: U.S. Patent Application 13/915,410, 2013. https://patents.google.com/patent/US20130329601.

[21] *Configuring OpenFlow*: https://techhub.hpe.com/eginfolib/networking/docs/switches/5940/5200-1028b_openflow_cg/content/491966856.htm Accessed 25 September 2022

[22] Farhady, H., Lee, H. Y., Nakao, A.: *Software-Defined Networking: A survey*. Comput. Net., 81 (2015) 79-95. https://doi.org/10.1016/j.comnet.2015.02.014

[23] Gude, N., Koponen, T., Pettit, J., Pfaff B., Casado M.: *NOX: towards an operating system for networks*, ACM 1574 SIGCOMM CCR, 38 (3) (2008), pp. 105–110, https://doi.org/10.1145/1384609.1384625

[24] *Jaxon*, 2012, https://bitbucket.org/user_localhost/jaxon/src/master/ Accessed 21 December 2021

[25] *OpenDaylight Controller Overview*, 2016, https://docs.opendaylight.org/en/latest/user-guide/opendaylight-controller-overview.html Accessed 21 December 2021

[26] *What is a Floodlight Controller?*, 2014, https://www.sdxcentral.com/networking/sdn/definitions/what-is-floodlight-controller/ Accessed 21 December 2021

[27] Erickson D.: *The Beacon OpenFlow Controller*, Stanford University, http://yuba.stanford.edu/~derickso/docs/hotsdn15-erickson.pdf Accessed 21 December 2021

[28] *IRIS*, 2013, http://openiris.etri.re.kr/ Accessed 21 December 2021

[29] *NodeFlow*, https://www.oreilly.com/library/view/software-defined-networking-with/9781783984282/a6393111-31f5-4de6-bc44-b926ac108e44.xhtml Accessed 21 December 2021

[30] *Helios*, 2012, https://nectoday.com/tag/helios/ Accessed 21 December 2021

[31] *Trema*, 2013, https://github.com/trema/trema Accessed 21 December 2021

[32] *POX Documentation*, 2015, https://noxrepo.github.io/pox-doc/ Accessed 21 December 2021

[33] Shin, M., Nam, K., Kim, H.: *Software-defined networking (SDN): A reference architecture and open APIs*, in: International Conference on ICT Convergence, ICTC 2012, 2012, pp. 360-361. http://doi.org/10.1109/ICTC.2012.6386859

[34] Al-Fares, M., Loukissas, A., Vahdat, A.: *A scalable, commodity data center network architecture.* ACM SIGCOMM Comput. Commun. Rev. 38 (4) (2008) 63-74. https://doi.org/10.1145/1402946.1402967

[35] Messaoud, S., Bradai, A., Moula, E.: *Online GMM Clustering and Mini-Batch Gradient Descent based Optimization for Industrial IoT 4.0*, IEEE Transactions on Industrial Informatics, 16 (2) (2020) 1427 - 1435, https://doi.org/10.1109/TII.2019.2945012

[36] Messaoud, S., Bradai, A., Ahmed, O. B., Anh Quang Ph. T., Atri, M., Hossain, M. Sh.: *Deep Federated Q-Learning-Based Network Slicing for Industrial IoT*, IEEE Transactions on Industrial Informatics, 17(8) (2021), 5572 - 5582, https://doi.org/10.1109/TII.2020.3032165

[37] Levin, D., Canini, M., Schmid, S., Schaffert, F., Feldmann A.: *Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks*, in: USENIX Annual Technical Conference, Jun 2014, pp. 333–345. https://www.usenix.org/conference/atc14/technical-sessions/presentation/levin. Accessed 30 March 2021

[38] Amin, R., Reisslein, M., Shah, N.: *Hybrid SDN Networks: A Survey of Existing Approaches*. IEEE Commun. Surv. Tutor. 20 (4) (2018) 3259-3306. http://dx.doi.org/10.1109/COMST.2018.2837161

[39] Rathee, S., Sinha, Y., Haribabu, K.: *A Survey: Hybrid SDN*. J. Netw. Comput. Appl. 100 (2017) 35-55. https://doi.org/10.1016/j.jnca.2017.10.003

[40] Vissicchio, S., Vanbever, L., Bonaventure, O.: *Opportunities and Research Challenges of Hybrid Software Defined Networks*. ACM SIGCOMM Comput. Commun. Rev. 44, (2) (2014). 70-75. https://doi.org/10.1145/2602204.2602216

[41] Vissicchio, S., Vanbever, L., Cittadini, L., Xie, G. G., Bonaventure, O.: *Safe routing reconfigurations with route redistribution*, in: IEEE Conference on Computer Communications, IEEE INFOCOM 2014. pp. 199-207. http://doi.org/10.1109/INFOCOM.2014.6847940

[42] He, J., Song, W.: *Achieving near-optimal traffic engineering in hybrid Software Defined Networks*, in: IFIP Networking Conference (IFIP Networking), 2015, pp. 1-9. http://doi.org/10.1109/IFIPNetworking.2015.7145321

[43] Bahnasse, A., Louhab, F.E., Ait Oulahyane, H., Talea, M., Bakali, A.: *Novel SDN architecture for smart MPLS Traffic Engineering-DiffServ Aware management*. Future Gener. Comput. Syst. 87 (2018) 115-126. https://doi.org/10.1016/j.future.2018.04.066

[44] Canini, M., Feldmann, A., Levin, D., Schaffert, F., Schmid, S.: *Software-defined networks: Incremental deployment with Panopticon*. IEEE Comput. 47 (11) (2014) 56–60. https://doi.org/10.1109/MC.2014.330

[45] Poularakis, K., Iosifidis, G., Smaragdakis, G., Tassiulas L.: *One step at a time: Optimizing SDN upgrades in ISP networks*. IEEE Conference on Computer Communications, IEEE INFOCOM 2017 (2017) pp. 1-9. https://doi.org/10.1109/INFOCOM.2017.8057136

[46] Xu, H., Fan, J., Wu, J., Qiao, C., Huang, L.: *Joint deployment and routing in hybrid SDNs*. IEEE/ACM 25th International Symposium on Quality of Service, IWQoS 2017 (2017) pp. 1–10. https://doi.org/10.1109/IWQoS.2017.7969133

[47] Casey, D. J., Mullins, B. E.: *SDN shim: Controlling legacy devices. IEEE 40th Conference on Local Computer Networks*, LCN 2015 (2015) pp. 169–172. https://doi.org/10.1109/LCN.2015.7366298

[48] Nunez-Martinez, J., Baranda, J., Mangues-Bafalluy, J.: *A service-based model for the hybrid software-defined wireless mesh backhaul of small cells*. 11th International Conference on Network and Service Management, CNSM 2015 (2015) pp. 390–393. https://doi.org/10.1109/CNSM.2015.7367388

[49] Balta, M., Özçelik, İ.: *A 3-stage fuzzy-decision tree model for traffic signal optimization in urban city via a SDN based VANET architecture*. Future Gener. Comput. 104 (2020) 142-158. https://doi.org/10.1016/j.future.2019.10.020

[50] Bhattacharyya, S., Katramatos, D., Yoo, S.: *Why wait? Let us start computing while the data is still on the wire. Future Gener. Comput*. Syst. 89 (2018) 563-574. https://doi.org/10.1016/j.future.2018.07.024

[51] Neghabi, A. A., Navimipour, N. J., Hosseinzadeh, M., Rezaee, A.: *Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature*. IEEE Access, 6 (2018) 14159 - 14178. https://doi.org/10.1109/ACCESS.2018.2805842

[52] Shang, Z., Chen, W., Ma, Q., Wu, B.: *Design and implementation of server cluster dynamic load balancing based on OpenFlow*. International Joint Conference on Awareness Science and Technology & Ubi-Media Computing, iCAST 2013 & UMEDIA 2013 (2013) pp. 691-697. https://doi.org/10.1109/ICAwST.2013.6765526

[53] Xu, Z., Huang, R., Bhuyan, L. N.: *Load balancing of DNS-based distributed Web server systems with page caching*. Tenth International Conference on Parallel and Distributed Systems, ICPADS 2004 (2004) pp. 587-594. https://doi.org/10.1109/ICPADS.2004.1316141

[54] Tong, R., Zhu, X.: *A Load Balancing Strategy Based on the Combination of Static and Dynamic*, in: 2nd International Workshop on Database Technology and Applications (2010) pp. 1-4. https://doi.org/10.1109/DBTA.2010.5658951

[55] Kumar, P., Kumar, R.: *Issues and Challenges of Load Balancing Techniques in Cloud Computing*, ACM Computing Surveys, 51(6), 2019, 1–35, https://10.1145/3281010

[56] Belgaum, M. R., Musa, S., Alam, M. M., Su'ud, M. M.: *A Systematic Review of Load Balancing Techniques in Software-Defined Networking*. IEEE Access, 8 (2020) pp. 98612–98636. https://10.1109/access.2020.2995849

[57] Hota, A., Mohapatra, S., Mohanti, S.: *Survey of Different Load Balancing Approach-Based Algorithms in Cloud Computing*: A Comprehensive Review, [Advances in Intelligent Systems and Computing] Computational Intelligence in Data Mining Volume, 711 (2019), pp. 99–110. https://10.1007/978-981-10-8055-5_10

[58] Ghomi, E. J., Rahmani, A. M., Qader, N. N.: *Load-balancing Algorithms in Cloud Computing: A Survey*, Journal of Network and Computer Applications, 88 (2017), pp. 50-71, https://10.1016/j.jnca.2017.04.007

[59] Thakur, A., Goraya, M. S.: *A TAXONOMIC SURVEY ON LOAD BALANCING IN CLOUD*, Journal of Network and Computer Applications, 98 (2017), pp. 43-57 https://10.1016/j.jnca.2017.08.020

[60] Kokilavani, T., Amalarethinam, D. I. G.: *Load balanced min-min algorithm for static meta-task scheduling in grid computing*, International Journal of Computer Applications, 20.2 (2011), pp. 43-49, https://doi.org/10.5120/2403-3197

[61] Bhoi, U., Ramanuj, P.N.: *Enhanced max-min task scheduling algorithm in cloud computing*, International Journal of Application or Innovation in Engineering and Management (IJAIEM), 2.4 (2013), pp. 259-264

[62] Mishra, S. K., Sahoo, B., Parida, P. P.: *Load Balancing in Cloud Computing: A big Picture*, Journal of King Saud University - Computer and Information Sciences, 32.2 (2020), pp. 149-158, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2018.01.003

[63] Hidayat, T., Azzery, Y., Mahardiko, R.: *Load balancing network by using round robin algorithm: a systematic literature review*, Jurnal Online Informatika, 4.2 (2020), pp. 85-89, https://10.15575/join.v4i2.446

[64] Wang, W., Casale, G.: *Evaluating Weighted Round Robin Load Balancing for Cloud Web Services*, 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2014, pp. 393-400. https://10.1109/SYNASC.2014.59

[65] Singh, G., Kaur, K.: *An improved weighted least connection scheduling algorithm for load balancing in web cluster systems*, International Research Journal of Engineering and Technology (IRJET), 5.3 (2018), p. 6.

[66] Guo, Z., Su, M., Xu, Y., Duan, Z., Wang, L., Hui, S., Chao, H. J.: *Improving the performance of load balancing in software-defined networks through load variance-based synchronization*. Comput. Netw. 68 (2014) 95-109. https://doi.org/10.1016/j.comnet.2013.12.004

[67] Yeganeh, S. H., Tootoonchian, A., Ganjali, Y.: *On scalability of software-defined networking*. IEEE Commun. Mag. 51 (2) (2013) 136-141. https://doi.org/10.1109/MCOM.2013.6461198

[68] Sharma, S., Singh, S., Sharma, M.: *Performance analysis of load balancing algorithms*. World Acad Sci Eng Technol. 38 (2008) 269-272. https://doi.org/10.5281/zenodo.1061232

[69] Kaur, S., Singh, J.: *Implementation of Server Load Balancing in Software Defined Networking*, in: S. Satapathy, J. Mandal, S. Udgata, V. Bhateja (Eds.) Information Systems Design and Intelligent Applications: Advances in Intelligent Systems and Computing, vol 434, Springer, New Delhi (2016). https://doi.org/10.1007/978-81-322-2752-6_14

[70] Song, P., Liu, Y., Liu, T., Qian, D.: *Flow Stealer: lightweight load balancing by stealing flows in distributed SDN controllers*. Sci. China Inf. Sci. 60 (3) (2017) 032202. https://doi.org/10.1007/s11432-016-0333-0

[71] Hu, Y., Luo, T., Beaulieu, N. C., Wang, W.: *An Initial Load-Based Green Software Defined Network*. App. Sci. 7 (5) (2017) 459. https://doi.org/10.3390/app7050459

[72] Zhang, J., Xi, K., Luo, M., Chao, H. J.: *Load balancing for multiple traffic matrices using SDN hybrid routing*. IEEE 15th International Conference on High Performance Switching and Routing, HPSR 2014 (2014) pp. 44-49. https://doi.org/10.1109/HPSR.2014.6900880

[73] Wang, R., Butnariu, D., Rexford, J.: *OpenFlow-based server load balancing gone wild*, in: 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services, Hot-ICE'11 (2011). https://www.usenix.org/legacy/events/hotice11/tech/full_papers/Wang_Richard.pdf. Accessed 30 March 2021

[74] Koerner, M., Kao, O.: *Multiple service load-balancing with OpenFlow*, in: IEEE 13th International Conference on High Performance Switching and Routing (2012) pp. 210-214. https://doi.org/10.1109/HPSR.2012.6260852

[75] Ukrist,S., Pradittasnee, L., and Kitsuwan, N.: *Resolving Load Imbalance State for SDN by Minimizing Maximum Load of Controllers*, Journal of Network and Systems Management 29 (4) (2021) 1-28. https://doi.org/10.1007/s10922-021-09612-w

[76] El Kamel, A., and Youssef, H.: *Improving Switch-to-Controller Assignment with Load Balancing in Multi-controller Software Defined WAN (SD-WAN)*, Journal of Network and Systems Management 28 (2020) 553–575. https://doi.org/10.1007/s10922-020-09523-2

[77] Guo, Y., Wang, Z., Yin, X., Shi, X., and Wu, J.: *Traffic engineering in SDN/OSPF hybrid network*, in International Conference on Network Protocols, 2014, pp. 563-568, https://doi.org/10.1109/ICNP.2014.90

[78] Ren, C., Wang,S., Ren, J., Wang, X., Song, T., and Zhang, D.: *Enhancing traffic engineering performance and flow manageability in hybrid SDN*, in IEEE Global Communication Conference, IEEE GLOBECOM 2016, pp. 1-7, https://10.1109/GLOCOM.2016.7841819

[79] Lin, C., Wang, K., and Deng, G.: *A QoS-aware routing in SDN hybrid networks*,Procedia Computer Science, 110 (2017), 242-249, https://doi.org/10.1016/j.procs.2017.06.091

[80] *EVE-NG*, https://www.eve-ng.net/ (2020). Accessed 30 March 2021

[81] *Open vSwitch*, https://www.openvswitch.org/ (2020). Accessed 30 March 2021

[82] Bojovic, P. D., Malbasic, T.: *SDN-LBORU - Load Balancing by Optimizing Resource Utilization*. GitHub repository. https://github.com/Paxy/SDN-LBORU (2020). Accessed 30 March 2021

[83] Chowdhury, M. Z., Shahjalal, M., Ahmed, S., Jang, Y.M.: *6GWireless Communication Systems: Applications, Requirements, Technologies, Challenges, and Research Directions*. IEEE Open J. Commun. Soc. 2020, 1, 957–975.

[84] Akyildiz, I.F., Kak, A., Nie, S.: *6G and Beyond: The Future of Wireless Communications Systems*. IEEE Access 2020, 8, 133995–134030.

[85] Rajatheva, N., Atzeni, I., Bjornson, E., Bourdoux, A., Buzzi, S., Dore, J.B., Erkucuk, S., Fuentes, M., Guan, K., Hu, Y., et al.: *White paper on broadband connectivity in 6G*. arXiv 2020, arXiv:2004.14247.

[86] Khan, L.U., Yaqoob, I., Tran, N.H., Han, Z., Hong, C.S.: *Network Slicing: Recent Advances, Taxonomy, Requirements, and Open Research Challenges*. IEEE Access 2020, 8, 36009–36028.

[87] Duan, Q., Wang, S., Ansari, N.: *Convergence of Networking and Cloud/Edge Computing: Status, Challenges, and Opportunities*. IEEE Netw. 2020, 34, 1–8.

[88] Marsch, P., Bulakci, Ö., Queseth, O., Boldi, M.: *E2E Architecture in 5G System Design: Architectural and Functional Considerations and Long Term Research*, 1st ed., JohnWiley Sons, Inc.: Hoboken, NJ, USA, 2018, pp. 81–115.

[89] Javed, F., Antevski, K., Mangues, J., Giupponi, L., Bernardos, C.J.: *Distributed Ledger Technologies For Network Slicing: A Survey*. IEEE Access 2022, 10, 19412–19442.

[90] Lebedenko, T., Yeremenko, O., Harkusha, S., Ali, A.S.: *Dynamic model of queue management based on resource allocation in telecommunication networks. In Proceedings of the TCSET*, Lviv-Slavske, Ukraine, 20–24 February 2018, pp. 1035–1038.

[91] *OpenFlow Configuration and Management Protocol OF-CONFIG 1.0*, ONF TS-004. 2011. Available online: https://opennetworking.org/wp-content/uploads/2013/02/of-config1dot0-final.pdf Accessed 5 February 2022

[92] *The Open vSwitch Database Management Protocol*, RFC 7047. 2013. Available online: https://datatracker.ietf.org/doc/html/ rfc7047.txt Accessed 5 February 2022

[93] Li, Y., Huang, J., Sun, Q., Sun, T., Wang, S.: *Cognitive Service Architecture for 6G Core Network*. IEEE Trans. Ind. Inform. 2021, 17, 7193–7203.

[94] Kaloxylos, A.: *A Survey and an Analysis of Network Slicing in 5G Networks*. IEEE Commun. Stand. Mag. 2018, 2, 60–65.

[95] Agyapong, P., Iwamura, M., Staehle, D., Kiess, W., Benjebbour, A.: *Design considerations for a 5G network architecture*. IEEE Commun. Mag. 2014, 52, 65–75.

[96] Bosshart, P., Gibb, G., Kim, H.S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., Horowitz, M.: *Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN*. ACM Sigcomm Comput. Commun. Rev. 2013, 43, 99–110.

[97] Banchs, A., Fiore, M., Garcia-Saavedra, A., Gramaglia, M.: *Network intelligence in 6G: Challenges and opportunities*. In Proceedings of the 16th ACMWorkshop on Mobility in the Evolving Internet Architecture, New Orleans, LA, USA, 25 October 2021, pp. 7–12.

[98] Taleb, T., Afolabi, I., Samdanis, K., Yousaf, F.Z.: *On multi-domain network slicing orchestration architecture and federated resource control*. IEEE Netw. 2019, 33, 242–252.

[99] Guan, W., Zhang, H., Leung, V.C.M.: *Customized Slicing for 6G: Enforcing Artificial Intelligence on Resource Management*. IEEE Netw. Early Access 2021, 35, 264–271.

[100] Vincenzi, M., Antonopoulos, A., Kartsakli, E., Vardakas, J., Alonso, L., Verikou, C.: *Multi-tenant slicing for spectrum management on the road to 5G*. IEEE Wirel. Commun. 2017, 24, 118–125.

[101] Abiko, Y., Saito, T., Ikeda, D., Ohta, K., Mizuno, T., Mineno, H.: *Flexible resource block allocation to multiple slices for radio access network slicing using deep reinforcement learning*. IEEE Access 2020, 8, 68183–68198.

[102] El Amri, A., Meddeb, A.: *Optimal server selection for competitive service providers in network virtualization context*. Telecommun. Syst. 2021, 77, 451–467.

[103] Zhang, Z., Xiao, Y., Ma, Z., Xiao, M., Ding, Z., Lei, X., Karagiannidis, G.K., Fan, P.: *6GWireless Networks: Vision, Requirements, Architecture, and Key Technologies*. IEEE Veh. Technol. Mag. 2019, 14, 28–41.

[104] Antonopoulos, A.: *Bankruptcy problem in network sharing: Fundamentals, applications and challenges*. IEEE Wirel. Commun. 2020, 27, 81–87.

[105] Ramrao, J.V., Jain, A.: *Dynamic 5G Network Slicing*. Int. J. Adv. Trends Comput. Sci. Eng. 2021, 10, 1006–1010.

[106] Abbas, K., Afaq, M., Ahmed Khan, T., Rafiq, A., Song, W.-C.: *Slicing the Core Network and Radio Access Network Domains through Intent-Based Networking for 5G Networks*. Electronics 2020, 9, 1710.

[107] 25. Kukli´nski, S., Tomaszewski, L., Kołakowski, R., Chemouil, P.: *6G-LEGO: A Framework for 6G Network Slices*. J. Commun. Netw. 2021, 23, 442–453.

[108] Chergui, H., Blanco, L., Garrido, L.A., Ramantas, K., Kukli´nski, S., Ksentini, A., Verikoukis, C.: *Zero-Touch AI-Driven Distributed Management for Energy-Efficient 6GMassive Network Slicing*. IEEE Netw. 2021, 35, 43–49.

[109] Foukas, X., Patounas, G., Elmokashfi, A., Marina, M.K.: *Network Slicing in 5G: Survey and challenges*. IEEE Commun. Mag. 2017, 55, 94–100.

[110] Biczok, G., Dramitinos, M., Toka, L., Heegaard, P.E., Lonsethagen, H.: *Manufactured by Software: SDN-Enabled Multi-Operator Composite Services with the 5G Exchange*. IEEE Commun. Mag. 2017, 55, 80–86.

[111] Afolabi, I., Taleb, T., Samdanis, K., Ksentini, A., Flinck, H.: *Network slicing and softwarization: A survey on principles, enabling technologies, and solutions*. IEEE Commun. Surv. Tutor. 2018, 20, 2429–2453.

[112] You, X., Wang, C.-X., Huang, J., Gao, X., Zhang, Z., Wang, M., Huang, Y., Zhang, C., Jiang, Y., Wang, J., et al.: *Towards 6G wireless communication networks: Vision, enabling technologies, and new paradigm shifts*. Sci. China Inf. Sci. 2020, 64, 110301.

[113] Ziegler, V., Yrjola, S.: *6G Indicators of Value and Performance.* In Proceedings of the 2nd 6GWireless Summit, Levi, Finland, 17–20 March 2020, pp. 1–5.

[114] Palma, D., Gonçalves, J., Sousa, B., Cordeiro, L., Simoes, P., Sharma, S., Staessens, D.: *The QueuePusher: Enabling Queue Management in OpenFlow*. In Proceedings of the Third European Workshop on Software Defined Networks, Budapest, Hungary, 1–3 September 2014, pp. 125–126.

[115] Kim,W., Sharma, P., Lee, J., Banerjee, S., Tourrilhes, J., Lee, S.J., Yalagandula, P.: *Automated and Scalable QoS Control for Network Convergence*. In Proceedings of the INM/WREN'10, San Jose, CA, USA, 27 April 2010, pp. 1–6.

[116] Durner, R., Blenk, A., Kellerer,W.: *Performance study of dynamic QoS management for OpenFlow-enabled SDN switches*. In Proceedings of the IWQoS, Portland, OR, USA, 15–16 June 2015, pp. 177–182.

[117] Kuzniar, M., Peresini, P., Kostic, D.: *What You Need to Know about SDN Control and Data Planes,* EPFL: Lausanne, Switzerland, 2014.

[118] Bozakov, Z., Rizk, A.: *Taming SDN controllers in heterogeneous hardware environments*. In Proceedings of the EWSDN, Berlin, Germany, 10–11 October 2013, pp. 50–55.

[119] Kuzniar, M., Peresini, P., Kostic, D.: *What you need to know about sdn flow tables*. In Proceedings of the PAM, New York, NY, USA, 19–20 March 2015, pp. 347–359.

[120] Mohan, P.M., Divakaran, D.M., Gurusamy, M.: *Performance study of TCP flows with QoS-supported OpenFlow in data center networks*. In Proceedings of the ICON, Singapore, 11–13 December 2013, pp. 1–6.

[121] Nguyen-Ngoc, A., Lange, S., Gebert, S., Zinner, T., Tran-Gia, P., Jarschel, M.: *Investigating isolation between virtual networks in case of congestion for a Pronto 3290 switch*. In Proceedings of the SDNFlex 2015, Cottbus, Germany, 9–13 March 2015, pp. 1–5.

[122] Jeong, S., Lee, D., Hyun, J., Li, J., Hong, J.-W.K.: *Application-aware traffic engineering in software-defined network.* In Proceedings of the APNOMS, Seoul, Korea, 27–29 September 2017, pp. 315–318.

[123] Agarwal, S., Kodialam, M.i., Lakshman, T.V.: *Traffic engineering in software defined networks*. In Proceedings of the INFOCOM, Turin, Italy, 14–19 April 2013, pp. 2211–2219.

[124] Huan, N.F., Liao, I.-J., Liu, H.-W., Wu, S.-J., Chou, C.-S.: *A dynamic QoS management system with flow classification platform for software-defined networks*. In Proceedings of the UMEDIA, Colombo, Sri Lanka, 24–26 August 2015, pp. 72–77.

[125] Baklizi, M.: *Weight Queue Dynamic Active Queue Management Algorithm*. Symmetry 2020, 12, 2077.

[126] Khan, S., Hussain, F.K., Hussain, O.K.: *Guaranteeing end-to-end QoS provisioning in SOA based SDN architecture: A survey and Open Issues*. Future Gener. Comput. Syst. 2021, 119, 176–187.

[127] Pedreno-Manresa, J.J., Khodashenas, P.S., Siddiqui, M.S., Pavon-Marino, P.: *Dynamic QoS/QoE Assurance in Realistic NFVEnabled 5G Access Networks*. In Proceedings of the ICTON, Girona, Spain, 2–6 July 2017, pp.1–4.

[128] Sonkoly, B., Gulyás, A., Németh, F., Czentye, J., Kurucz, K., Novák, B., Vaszkun, G.: *On QoS Support to Ofelia OpenFlow*. In Proceedings of the EuropeanWorkshop on Software Defined Networking, Darmstadt, Germany, 25–26 October 2012, pp. 109–113.

[129] Bari, M.F., Chowdhury, S.R., Ahmed, R., Boutaba, R. PolicyCop: *An Autonomic QoS Policy Enforcement Framework for Software Defined Networks*. In Proceedings of the SDN4FNS, Trento, Italy, 11–13 November 2013, pp. 1–7.

[130] Bueno, I., Aznar, J.I., Escalona, E., Ferrer, J., Garcia-Espin, J.A.: *An OpenNaaS based SDN framework for dynamic QoS control*. In Proceedings of the SDN4FNS, Trento, Italy, 11–13 November 2013, pp. 1–7.

[131] Panev, S., Latkoski, P.: *Modeling of OpenFlow-related handover messages in mobile networks*. Telecommun. Syst. 2020, 75, 307–318.

[132] Llorens-Carrodeguas, A., Leyva-Pupo, I., Cervelló-Pastor, C., Piñeiro, L., Siddiqui, S.: *An SDN-based Solution for Horizontal Auto-Scaling and Load Balancing of Transparent VNF Clusters*. Sensors 2021, 21, 8283.

[133] Rost, P., Mannweiler, C., Michalopoulos, D.S., Sartori, C., Sciancalepore, V., Sastry, N., Bakker, H.: *Network Slicing to Enable Scalability and Flexibility in 5G Mobile Networks*. IEEE Commun. Mag. 2017, 55, 72–79.

[134] Balan, D.G., Potorac, D.A.: *Linux HTB queuing discipline implementations*. In Proceedings of the First International Conference on Networked Digital Technologies, Ostrava, Czech Republic, 29–31 July 2009, pp. 122–126.

[135] *DynQoS* [Internet]. GitHub: Petar Bojovic. Available online https://github.com/Paxy/DynQoS Accessed 5 February 2022.

[136] Avallone, S., Guadagno, S., Emma, D., Pescape, A., Ventre, G.: *D-ITG distributed internet traffic generator.* In Proceedings of the QEST 2004, Enschede, The Netherlands, 27–30 September 2004, pp. 316–317.

[137] *Bandwidth Monitor NG* [Internet]. GitHub: Volker Gropp. Available online: https://github.com/vgropp/bwm-ng Accessed 5 February 2022.

11. Prilog – detalji implementacije predloženog rešenja za balansiranje opterećenja servera

U ovom prilogu opisana je implementacija programskih rešenja:

1. modula za balansiranje opterećenja i
2. modula za praćenje opterećenja server mašina.

11.1 Implementacija modula za balansiranje opterećenja

Programsko rešenje modula za balansiranje opterećenja (*SDN-LBORU - Load Balancing by Optimizing Resource Utilization, GitHub repository*: *https://github.com/Paxy/SDN-LBORU*) napisano je u programskom jeziku *Python*. Za samu realizaciju modula za balansiranje opterećenja u velikoj meri primenjen je *POX* SDN kontroler (pisan u jeziku *Python*).

Na samom početku programa izvršen je uvoz *core* objekta *POX* kontrolera, kao i niz njegovih biblioteka potrebnih za dalji razvoj programskog rešenja. Definisan je i objekat za praćenje izvršavanja programa u konzoli metodom *getLogger()*.

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.addresses import IPAddr, EthAddr
from pox.lib.revent import EventHalt
from pox.lib.packet.arp import arp
from pox.lib.packet.ethernet import ethernet

log = core.getLogger("f.t_p")
```

U narednom koraku definišu se globalne konstante:

```
virtual_ip = IPAddr("192.168.100.222")
virtual_mac = EthAddr("00:50:00:00:0d:00")
server = {}
server[0] = {'ip':IPAddr("192.168.100.248"), 'mac':EthAddr("00:50:00:00:06:00")}
server[1] = {'ip':IPAddr("192.168.100.247"), 'mac':EthAddr("00:50:00:00:07:00")}
```

Ove konstante se odnose na IP i MAC adrese servera. Postoji samo jedan par virtuelnih IP i MAC adresa, jer njih dele svi fizički (realni) serveri. U ranije opisanom *testbed* okruženju nalaze se dva servera, te je potrebno definisati dva para fizičkih IP i MAC adresa (u gornjem kodu: *server[0]* i *server[1]*).

Funkcionalnosti modula su implementirane sa objektno-orijentisanim pristupom, odnosno definisanjem klasa. Modul razlikuje dve klase:

1. *Switch*
2. *proactive_flow*

Inicijalizacijom objekta klase *Switch* uspostavlja se veza između SDN sviča i SDN kontrolera. Pored toga, inicijalizacijom se poziva metoda *addListeners()*, zadužena za prisluškivanje novih događaja na sviču.

```
class Switch(object):
  def __init__(self, connection):
    self.connection = connection
    connection.addListeners(self)
```

Nakon toga sledi definicija funkcije *_handle_PacketIn()*. Ovom metodom se rukuje ARP paketima koji pristižu na kontroler sa sviča, kao i paketima koji spadaju u kategoriju reaktivnih tokova (6.1.3). Najpre se paket nekog događaja parsira, a zatim se ispituje njegov sadržaj i ako paket odgovara ARP paketu čija je destinaciona adresa virtuelna IP adresa ili jednom od dva pravila reaktivnih tokova definisanih u 6.1.3, paket se nakon odgovarajuće obrade prosleđuje nazad sviču.

```
def _handle_PacketIn(self, event):
  packet = event.parsed

  # ARP handling
  if packet.type == 0x0806:
    if packet.payload.opcode == arp.REQUEST:
      if packet.payload.protodst == virtual_ip:
```

```
        # form a reply packet
        arp_reply = arp()
        arp_reply.hwsrc = virtual_mac
        arp_reply.hwdst = packet.src
        arp_reply.opcode = arp.REPLY
        arp_reply.protosrc = virtual_ip
        arp_reply.protodst = packet.payload.protosrc
        ether = ethernet()
        ether.type = ethernet.ARP_TYPE
        ether.dst = packet.src
        ether.src = virtual_mac
        ether.payload = arp_reply

        # send this packet to the switch
        packet_out = of.ofp_packet_out()
        packet_out.data=ether.pack()
        packet_out.actions.append(of.ofp_action_output(port=of.OFPP_TABLE))
        event.connection.send(packet_out)

# Handle traffic destined to virtual IP
if packet.type == 0x0800:
  if packet.payload.dstip == virtual_ip:

    # implement load balancing algorithm here

    # preparing packet for server
    msg = of.ofp_flow_mod()
    msg.priority = 1500
    msg.match = of.ofp_match()
    msg.match.dl_type = 0x0800
    msg.match.nw_proto = 6
    msg.match.nw_dst = virtual_ip
    msg.match.nw_src = packet.payload.srcip
    msg.match.tp_src = packet.payload.payload.srcport
    msg.actions.append(of.ofp_action_dl_addr(
              of.OFPAT_SET_DL_DST,
              selected_server_mac))
    msg.actions.append(of.ofp_action_nw_addr(
              of.OFPAT_SET_NW_DST,
              selected_server_ip))
    msg.actions.append(of.ofp_action_output(
              port=of.OFPP_NORMAL))
    event.connection.send(msg)

    # preparing packet for client
    rev_msg = of.ofp_flow_mod()
    rev_msg.priority = 1500
```

```
        rev_msg.match = of.ofp_match()
        rev_msg.match.dl_type = 0x0800
        rev_msg.match.nw_proto = 6
        rev_msg.match.nw_src = selected_server_ip
        rev_msg.match.nw_dst = msg.match.nw_src
        rev_msg.match.tp_dst=msg.match.tp_src
        rev_msg.actions.append(of.ofp_action_dl_addr(
                    of.OFPAT_SET_DL_SRC,
                    virtual_mac))
        rev_msg.actions.append(of.ofp_action_nw_addr(
                    of.OFPAT_SET_NW_SRC,
                    virtual_ip))
        rev_msg.actions.append(of.ofp_action_output(
                    port=of.OFPP_NORMAL))
        event.connection.send(rev_msg)

    return EventHalt
```

U prethodnom kodu, prilikom upravljanja paketom koji za destinacionu IP adresu ima virtuelnu IP adresu, neophodno je zameniti virtuelnu IP adresu sa realnom adresom servera. Zbog toga je na mestu gde je ostavljen komentar *# implement load balancing algorithm here*, neophodno odlučiti se za konkretan fizički server, primenom odgovarajućeg algoritma za datu metodu balansiranja opterećenja na serverima.

Kad je u pitanju RR mehanizam, kod za odlučivanje o serveru nije zahtevao pisanje posebnih funkcija, jer je dovoljno jednostavan i nije ga bilo potrebno izdvajati iz same *_handle_PacketIn()* metode:

```
index = server_index % total_servers
selected_server_ip = server[index]['ip']
selected_server_mac = server[index]['mac']
server_index += 1
```

U prethodnom kodu se nalaze dve pomoćne vrednosti, brojač *server_index* i konstanta *total_servers*. Prva je predviđena za pamćenje rednog broja poslednje server mašine koja je odabrana za opsluživanje korisnika, dok je u drugoj upisan broj servera koji su na raspolaganju metodi za balansiranje. Nad ove dve vrednosti se vrši matematička operacija deljenja sa ostatkom, i na taj način se dobija redni broj server mašine koja će biti odabrana. Posle odabira mašine neophodno je povećati vrednost brojača *server_index* za jedan, kako bi u narednoj iteraciji algoritma bio odabran sledeći na redu server.

LBBSRT i LBORU mehanizmi imaju poseban kod za odlučivanje o serveru, njihovi mehanizmi odlučivanja su izmešteni iz *_handle_PacketIn()* metode. Ovi mehanizmi su deo modula za nadzor opterećenja, i biće kasnije opisani. *_handle_PacketIn()* metodi se samo prosleđuje već doneta odluka:

```
selected_server_ip = IPAddr(ip_decision)
selected_server_mac = EthAddr(servers[ip_decision])
```

Na kraju *_handle_PacketIn()* metode, nalazi se povratna vrednost *EventHalt*.

Klasa *proactive_flow* sadrži metodu *_handle_openflow_ConnectionUp()*. Ova metoda je predviđena za momentalno slanje instrukcija predefinisanih (proaktivnih) tokova pri početku rada *POX* kontrolera. Na samom početku metode proverava se da li je uspostavljena veza sa SDN kontrolerom, i ako jeste, najpre se sviču šalju instrukcije za brisanje eventualno postojeće tabele tokova, a zatim se šalju instrukcije predefinisanih tokova definisanih u 6.1.1. Po završetku upisa ovih pravila u tabelu tokova na sviču, ostaje još da se instancira objekat klase *Switch*, čime na kontroleru počinje praćenje paketa koji su od interesa reaktivnim tokovima.

Klasa *proactife_flow* i njena metoda *_handle_openflow_ConnectionUp()* su date sledećim kodom:

```
class proactive_flow(object):
  def __init__ (self):
    self.log = log.getChild("Unknown")
    core.listen_to_dependencies(self, listen_args={'openflow':{'priority':0}})

  def _handle_openflow_ConnectionUp(self, event):
    if event.connection is None:
      self.log.debug("can't send table: disconnected")
      return

    # clear previous flows entries if any
    clear = of.ofp_flow_mod(command=of.OFPFC_DELETE)
    event.connection.send(clear)
    event.connection.send(of.ofp_barrier_request())

    # ARP -> PacketIn, Normal, Controller
    arp_rule = of.ofp_flow_mod()
    arp_rule.match = of.ofp_match()
    arp_rule.match.dl_type = 0x0806
    arp_rule.actions.append(of.ofp_action_output(
                port=of.OFPP_CONTROLLER))
    arp_rule.actions.append(of.ofp_action_output(
                port=of.OFPP_NORMAL))
```

```
    event.connection.send(arp_rule)

    # DstIP: vIP -> PacketIn, PRI: 1000
    vip_rule = of.ofp_flow_mod()
    vip_rule.match = of.ofp_match()
    vip_rule.match.dl_type = 0x0800
    vip_rule.match.nw_dst = virtual_ip
    vip_rule.priority = 1000
    vip_rule.actions.append(of.ofp_action_output(
                port=of.OFPP_CONTROLLER))
    event.connection.send(vip_rule)

    # Any -> Normal, PRI: 1001
    any_rule = of.ofp_flow_mod()
    any_rule.priority = 500
    any_rule.actions.append(of.ofp_action_output(
                port=of.OFPP_NORMAL))
    event.connection.send(any_rule)

    # initialize Switch() instance
    Switch(event.connection)
```

Važna napomena pri primeni *_handle_openflow_ConnectionUp()* metode prilikom testiranja LBBSRT i LBORU algoritma, jeste da je neophodan još jedan segment koda koji pokreće *daemon* nit modula za nadzor opterećenja server mašina. Neophodno je *Thread* klasi proslediti naziv glavne funkcije modula za praćenje stanja servera, predviđene za datu nit. Taj segment koda za LBBSRT metodu izgleda ovako:

```
t = Thread(target=ping_thread_func)
t.daemon = True
t.start()
```

gde je glavna funkcija praćenja stanja *servera ping_thread_func()*.

Dok za LBORU metodu isti kod ima sledeći sadržaj:

```
t = Thread(target=snmp_thread_func)
t.daemon = True
t.start()
```

gde je glavna funkcija praćenja stanja servera *snmp_thread_func()*.

Ovaj segment se ubacuje na mesto neposredno pre slanja instrukcije za brisanje postojećih tabela koje sadrže instrukcije tokova.

Modul balansiranja opterećenja se završava definicijom funkcije *launch()*, neophodne za registrovanje nove komponente *POX* kontrolera. Pri pokretanju glavnog programa *POX* kontrolera ova funkcija omogućava da se prethodno opisane funkcionalnosti modula puste u rad:

```
def launch():
  core.registerNew(proactive_flow)
```

Samo pokretanje glavnog programa vrši se iz komandne linije i to na sledeći način (LBORU metoda):

```
pox/pox.py log.level --DEBUG SNMP_LB
```

pri čemu je *SNMP_LB.py* naziv *Python* skripte, koja implementira program za balansiranje opterećenja LBORU metodom.

11.2 Implementacija modula za praćenje opterećenja server mašina

Za potrebe pisanja programskog rešenja modula za nadzor opterećenja (*SDN-LBORU - Load Balancing by Optimizing Resource Utilization, GitHub repository*: *https://github.com/Paxy/SDN-LBORU*), takođe je primenjen jezik *Python*. U prethodnom odeljku, pomenuta je globalna promenljiva *ip_decision*. Njena uloga jeste da neprestano čuva podatak o serveru, koji je datom metodom balansiranja odabran kao najpogodniji za prihvatanje korisničkih zahteva. Prethodno, bilo je reči o glavnim funkcijama modula za praćenje opterećenja (*ping_thread_func, snmp_thread_func*). Ove funkcije periodično osvežavaju podatak o optimalnom serveru za prihvatanje novih zahteva korisnika, a sada se naglašava da je njihova uloga da kontinuirano upisuju podatak o najpogodnijem serveru u promenljivu *ip_decision*.

Za potrebe testiranja rešenja, bilo je potrebno pored LBORU varijante ovog modula, realizovati i varijantu za LBBSRT metodu. Pošto je LBBSRT metoda nešto jednostavnija od LBORU, ona će biti izložena prva, a potom će biti predstavljena i sama LBORU metoda.

11.2.1 Realizacija programskog rešenja LBBSRT metode za balansiranje opterećenja

Pre početka dopune rešenja modulom za LBBSRT metodu, bilo je potrebno izvršiti uvoz nekoliko dodatnih (već postojećih) *Python* modula. Klasa *Thread* (već ranije pomenuta) pripada modulu *threading*:

```
from threading import Thread
```

Potom je uvedena funkcija *ping()*. Na osnovu njenog argumenta koji predstavlja IP adresu realnog servera, ona istovremeno testira dostupnost istog, i meri vreme potrebno poruci poslatoj serveru da se vrati nazad svom pošiljaocu (SDN kontroleru). U daljem tekstu ovo vreme će biti označavano sa RTT. Uvoz funkcije *ping():*

```
from ping3 import ping
```

Iz modula *statistics* bila je potrebna funkcija *stdev()* za računanje standardne devijacije (izraz (3)):

```
from statistics import stdev
```

Postoje dve globalne konstante *samples* i *treshold*:

```
samples = 20
treshold = 0.0001
```

Vrednost konstante *samples* predstavlja maksimalan broj uzoraka RTT vremena koji će biti posmatran u datom trenutku za svaki od servera. Prilikom ranijeg detaljnog objašnjenja LBBSRT metode, ova vrednost je bila označena sa *m*. Konstanta *treshold* je ranije opisan parametar λ LBBSRT metode. Optimalne vrednosti obe konstante bile su utvrđene i zadate prema mogućnostima *testbed* okruženja.

Za potrebe rukovanja *ping* alata, ali i zapisivanja vrednosti izmerenih ovim alatom u RAM memoriju, kao i za izračunavanja vrednosti standardne devijacije uzoraka RTT parametra, napisana je klasa *Pinger*:

```
class Pinger:
  def __init__(self, target, samples):
    self.target = target
    self.samples = samples
    self.data = list()

  def pingRound(self):
    t = ping(self.target)
    if t is None:
      return None
    self.add(t)
    return t

  def add(self, time):
    self.data.insert(0, time)
    if len(self.data) > self.samples:
      self.data.pop()

  def stdev(self):
    if len(self.data) < 2:
      return 0
    return stdev(self.data)
```

Konstruktor klase *Pinger* poseduje tri atributa:

1. *target* – ciljanu IP adresu servera,
2. *samples* – maksimalan broj uzoraka izmerenog RTT vremena i
3. *data* – lista u kojoj se čuvaju uzorci izmerenog RTT vremena.

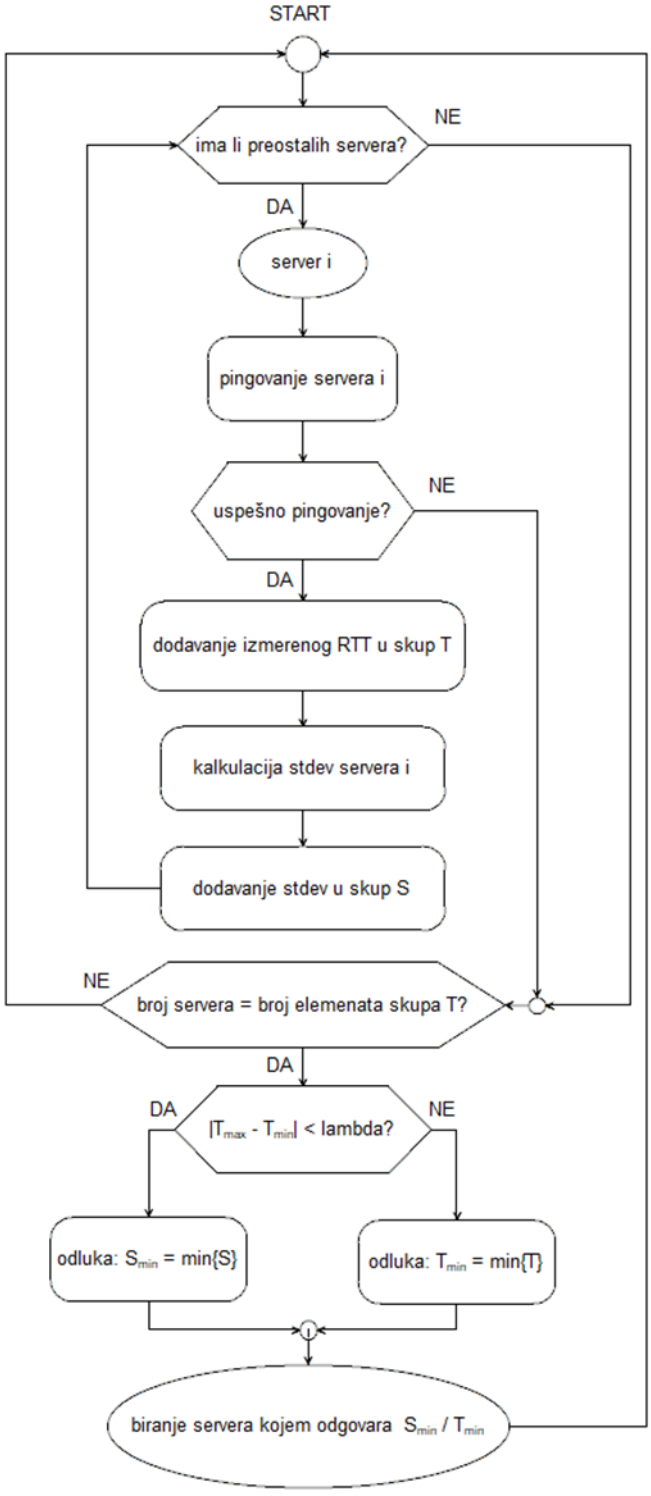
Metoda *pingRound()* implementira funkcionalnosti *ping* alata, od kojih je najbitnija merenje RTT vremena. Metoda *add()* dodaje izmerene uzorke u listu *data* i zadužena je da obezbedi da ta lista sadrži u svakom trenutku najviše onoliko uzoraka koliko je definisano sa *samples* atributom. Poslednja metoda *stdev()* izračunava standardnu devijaciju uzoraka za datu server mašinu.

Klasa *Pinger* je primenjena u glavnoj funkciji *ping_thread_func()*, koja predstavlja sponu između modula za balansiranje i modula za nadzor opterećenja. Na samom početku ove funkcije nalazi se lista IP adresa realnih servera. Potom sledi *for* petlja koja instancira svaki od objekata klase *Pinger*, dakle, u njoj se pri svakoj njenoj iteraciji kreira po jedan objekat klase *Pinger*, predviđen za svaki od servera. Nakon toga sledi beskonačna *while* petlja, koja implementira sam algoritam LBBSRT metode. Potrebno je skrenuti pažnju na to da je program *while* petlje adaptiran za *testbed* okruženje korišćeno u ovom radu, odnosno bilo bi ga potrebno izmeniti ukoliko bi se u *testbed* okruženju našlo više od dve server mašine.

```
def ping_thread_func():
  global treshold
  global samples
  global ip_decision

  server_list = ['192.168.100.247', '192.168.100.248']

  ping_objects = list()
  for server in server_list:
    ping_objects.append(Pinger(server, samples))

  while True:
    current=list()
    stdev_data = list()
    for server in ping_objects:
      ping = server.pingRound()
      if ping is None:
        break
      current.append(ping)
      stdev_data.append(server.stdev())

    if len(current) != len(ping_objects):
      continue

    diff = abs(max(current) - min(current))
    if diff < treshold:
      winner = stdev_data.index(min(stdev_data))
    else:
      winner = current.index(min(current))
    ip_decision = server_list[winner]
```

Svrha *while* petlje jeste da u svakoj svojoj iteraciji osvežava vrednost globalne promenljive *ip_decision*. S obzirom na to da je ova petlja smeštena u funkciji *ping_thread_func()*, koja se prosleđuje *daemon* niti, program *while* petlje se izvršava paralelno sa programom modula za balansiranje opterećenja.

Na sledećem dijagramu, prikazan je LBBSRT algoritam *while* petlje. Pošto je *while* petlja beskonačna, kod naveden u njenom telu se beskonačno ponavlja i na taj način se vrši neprestano osvežavanje podatka potrebnog za izbor servera sa najmanjim opterećenjem (promenljiva *ip_decision*).



Slika 11.1 LBBSRT algoritam *while* petlje

11.2.2 Realizacija programskog rešenja LBORU metode za balansiranje opterećenja

LBORU metoda zahteva kontinualan nadzor opterećenja, te je i za njene potrebe bilo potrebno uvesti primenu *Thread* klase komandom *import* kao i kod prethodne metode.

Pored *Thread* klase, bila su potrebna još tri *Python* paketa otvorenog koda. U pitanju su *time*, *pysnmp* i *numpy*, a njihov uvoz je izvršen na sledeći način:

```
from time import sleep
from pysnmp import hlapi
from numpy import array, amin
```

Iz paketa *time* je neophodna funkcija *sleep()* kojom se navodi period ponavljanja algoritma za nalaženje optimalnog servera koji će opslužiti datog korisnika.

Paket *pysnmp* sadrži API za rukovanje funkcionalnostima SNMP protokola - *hlapi*.

Paket *numpy* sadrži različite korisne matematičke funkcije kojima su optimizovane matematičke operacije nad podacima prikupljenim SNMP protokolom, a koje LBORU metoda koristi za donošenje odluke o izboru servera.

Kad je u pitanju odluka o izboru servera, ponovo je za pamćenje ovog podatka poslužila globalna promenljiva *ip_decision*.

Funkcionalnost pribavljanja podataka o zauzetosti resursa SNMP protokolom realizovana je kroz implementaciju klase *SNMPGetter*. Njen konstruktor izgleda ovako:

```
class SNMPGetter:
  def __init__(self, target, oids_list, credentials='public',
         port=1024):
    self.target = target
    self.oids_list = oids_list
    self.credentials = hlapi.CommunityData(credentials)
    self.port = port
    self.engine = hlapi.SnmpEngine()
    self.context = hlapi.ContextData()

    self.object_types = list()
    for oid in self.oids_list:
      self.object_types.append(hlapi.ObjectType(hlapi. ObjectType (oid)))
```

Ulazni parametri konstruktora su:

1. *target*
2. *oids_list*
3. *credentials*
4. *port*

Ulazni parametar konstruktora *target* je IP adresa konkretne server mašine s kojom se pokušava komunicirati SNMP protokolom. Parametar *oid_list* predstavlja listu svih OID identifikatora čije odgovarajuće vrednosti treba dobaviti SNMP protokolom. Parametar *credentials* se koristi za prosleđivanje metode verifikacije prava pristupa. Pošto se koristila verzija protokola SNMPv2, koja ne zahteva posebnu verifikaciju, vrednost ovog parametra postavljena je na *„public"* (javno). Moguće je definisati i *port* preko kojeg se želi ostvariti komunikacija sa serverom.

Što se tiče atributa klase *SNMPGetter*, oni praktično preuzimaju vrednosti ulaznih parametara. Izuzeci bi bili atributi *engine* i *context*, koji nisu od posebnog značaja za izradu programskog rešenja, već su tu samo kako bi definicija klase za SNMP komunikaciju bila potpuna.

Na kraju konstruktora potrebno je dodati sve OID identifikatore od interesa u jednu listu. Za ovakav postupak je potrebno primeniti dve klase *hlapi* paketa: *ObjectType* i *ObjectTypem*, u suprotnom *get()* metoda nije u mogućnosti da dobavi tražene podatke.

Klasa *SNMPGetter* poseduje tri metode:

1. *cast()*
2. *fetch()*
3. *get()*

Metoda *cast()* je jednostavna, ona samo konvertuje tip podataka dobavljenih SNMP protokolom u standardne *Python* tipove podataka: *int*, *float* i *string*:

```
def cast(self, value):
  try:
    return int(value)
  except (ValueError, TypeError):
    try:
      return float(value)
    except (ValueError, TypeError):
      try:
        return str(value)
      except (ValueError, TypeError):
        pass
  return value
```

Metoda *fetch()* implementirana je sledećim kodom:

```
def fetch(self, handler):
  result = None
  error_indication, error_status,
  error_index, var_binds = next(handler)
  if not error_indication and not error_status:
    items = dict()
    for var_bind in var_binds:
      items[str(var_bind[0])] = self.cast(var_bind[1])
    result = items
  else:
    raise RuntimeError('Got SNMP error: {0}'.format(error_indication))

   return result
```

Ukoliko se desi bilo kakva greška sa rukovanjem SNMP komunikacijom, *fetch()* metoda podiže odgovarajući izuzetak (*RuntimeError*). U slučaju da sve prođe očekivano, metoda će vratiti rečnik popunjen traženim vrednostima koje odgovaraju OID identifikatorima. Rečnik se koristi za skladištenje podataka, jer je potrebno vratiti više vrednosti OID identifikatora, te su ključevi ovog rečnika sami OID identifikatori, a vrednosti tih ključeva su same vrednosti odgovarajućih OID identifikatora. Može se primetiti da se prethodna *cast()* metoda primenjuje pri slaganju vrednosti OID identifikatora u rečnik.

Treća metoda, *get()*, definiše objekat rukovanja aplikativnom programskom spregom i potom poziva metodu *fetch()*, pri čemu rezultat izvršavanja metode *fetch()* predstavlja povratnu vrednost metode *get():*

```
def get(self):
  get_handler = hlapi.getCmd(
    self.engine,
    self.credentials,
    hlapi.UdpTransportTarget((self.target, self.port), timeout=10.0),
    self.context,
    *self.object_types
  )
return self.fetch(get_handler)
```

Kao poslednji parametar poziva funkcije *getCmd()* prosleđuje se **self.object_types*, koji je prethodno u konstruktoru klase *SNMPGetter* bio popunjen željenim OID identifikatorima. Na osnovu toga, *fetch()* metoda ima uvid u OID identifikatore koje treba da pretraži i dobavi.

Klasa *SNMPGetter* nalazi svoju primenu u glavnoj funkciji *snmp_thread_func()*. Ovo je ključna funkcija za komunikaciju između modula za nadzor i modula za balansiranje opterećenja. Na početku funkcije definisane su tri lokalne liste:

1. *server*_list – lista IP adresa realnih servera,
2. *oid_list* – lista svih OID identifikatora i
3. *weight_coeffs* - lista težinskih faktora (koeficijenata) za svaku od vrednosti OID identifikatora.

Prva lista sadrži IP adrese svih servera u *testbed* okruženju (u ovom slučaju 2).

Druga lista sadrži OID identifikatore resursa čije se opterećenje prati. U slučaju eksperimenata obavljenih u ovom radu, ova lista sadrži sledećih 5 OID identifikatora:

1. procenat procesorskog vremena,
2. količina podataka pročitanih sa diska,
3. količina podataka upisanih na disk,
4. količina podataka preuzetih sa *Downlink*-a i
5. količina podataka poslatih na *Uplink*.

Treća lista je predviđena za definisanje težinskih faktora (koeficijenata) za svaki od OID identifikatora iz prethodne liste. Ovaj koeficijent praktično diktira važnost svog odgovarajućeg parametra opterećenja u odnosu na druge parametre opterećenja.

Nakon definicija ovih listi sledi *for* petlja koja instancira svaki od objekata klase *SNMPGetter*. Ona ima zadatak da pri svakoj svojoj iteraciji kreira po jedan objekat klase *SNMPGetter*, predviđen za svaki od servera.

Kao i kod prethodnog algoritma (LBBSRT) ključnu ulogu ima beskonačna *while* petlja u kojoj je smešten sam algoritam LBORU metode. Međutim, ovaj segment koda nije strogo vezan za *testbed* okruženje kao što je to bio slučaj sa LBBSRT algoritmom. Dakle, program bi se mogao primeniti i na varijacije *testbed* okruženja u kojima bi pri raspodeli opterećenja učestvovalo i više od dva servera. Kako bi se rešenje proširilo na primenu više servera, samo je potrebno dopisati nove IP adrese u listu *server_list*.

```
def snmp_thread_func():
  global ip_decision

  server_list = ['192.168.100.247', '192.168.100.248']

  oid_list = ['1.3.6.1.2.1.25.3.3.1.2.196608',
        '1.3.6.1.4.1.2021.13.15.1.1.5.9',
        '1.3.6.1.4.1.2021.13.15.1.1.6.9',
        '1.3.6.1.2.1.2.2.1.10.2',
        '1.3.6.1.2.1.2.2.1.16.2']

  weight_coeffs = [1, 1, 0.25, 0.5, 0.5]

  snmp_objects = list()
```

```
for server in server_list:
  snmp_objects.append(SNMPGetter(server, oid_list))

while True:
  metrics = dict()
  for server in snmp_objects:
    snmp_results = server.get()
    if len(snmp_results) == len(oid_list):
      metrics[server.target] = list()
      for item in oid_list:
        metrics[server.target].append(snmp_results[item])
    else:
      continue

  ip_list = list()
  oid_values_list = list()
  for ip, oid_values in metrics.iteritems():
    ip_list.append(ip)
    oid_values_list.append(oid_values)

  a_array = array(oid_values_list)
  minimums = amin(a_array, axis=0).tolist()
  b = [[weight_coeffs[i] if parameters[i] == minimums[i] else 0
     for i in len(oid_list)] for parameters in oid_values_list]
  score = [sum(candidate) for candidate in b]
  winner_index = score.index(max(score))
  ip_decision = ip_list[winner_index]
  sleep(1)
```

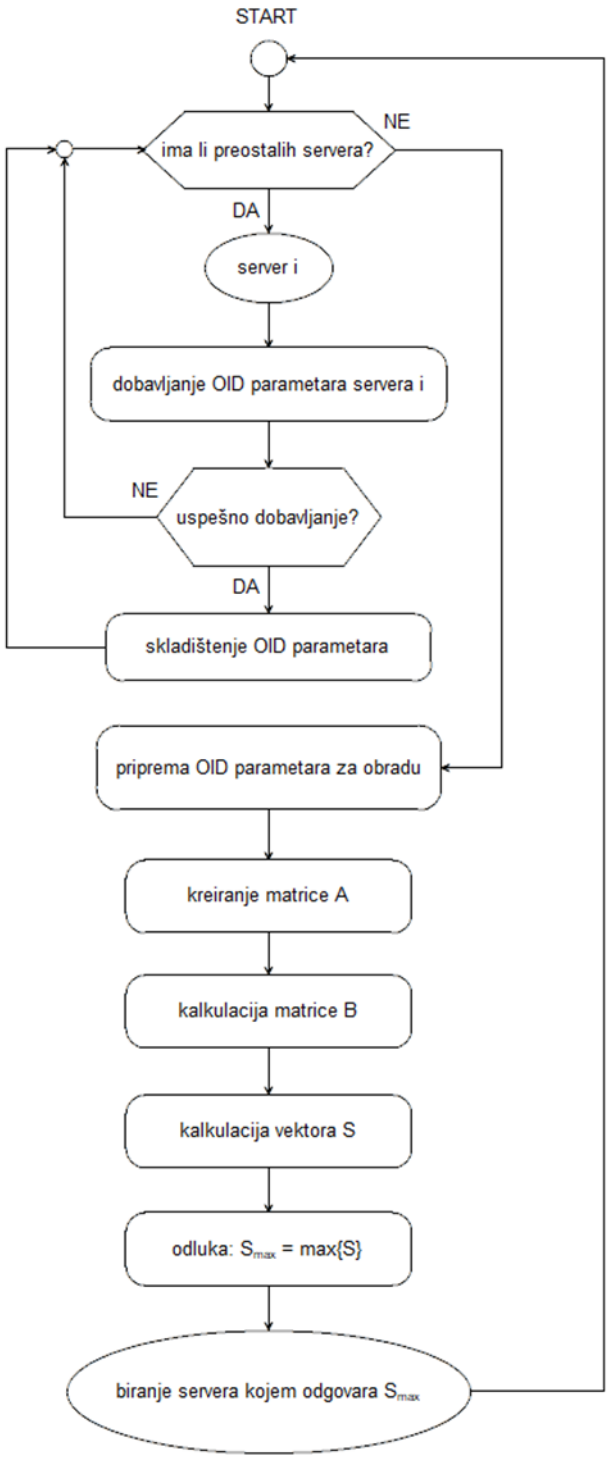
Prilagodljivost programskog rešenja ovog algoritma se ogleda u nekoliko njegovih modifikacija u odnosu na gore izložen kod:

1. Već je pomenuta modifikacija *server_ip* liste kako bi se okruženje proširilo sa više server mašina;
2. Listu *oid_list* je moguće proširiti ili smanjiti nužnim parametrima opterećenja server mašina. Samo je potrebno dopisati nove, odnosno izbrisati njene elemente. Dakle, ne postoji ograničenje po pitanju broja OID indentifikatora koji se mogu definisati. To znači da je moguće pratiti neograničen broj parametara opterećenja resursa server mašina. Obratiti pažnju na to da ukoliko se promeni *oid_list*, neophodno je na odgovarajući način modifikovati i *weight_coeffs* listu;
3. *weight_coeffs* lista se modifikuje u odnosu na zahteve samog servisa. Ukoliko je, na primer, za dati servis najbitnija što brža obrada podataka, prvi koeficijent liste *weight_coeffs* bi trebalo da ima veću vrednost od svih ostalih koeficijenata;

4. Moguće je i produžiti ili skratiti period ponavljanja koda beskonačne *while* petlje izmenom parametra funkcije *sleep()*. Ovim se praktično saopštava programu koliko često da osvežava vrednost promenljive *ip_decision*.

Kao i pre, svrha *while* petlje je da u svakoj svojoj iteraciji osvežava vrednost globalne promenljive *ip_decision*. Petlja je smeštena u funkciju *snmp_thread_func()*, koja se prosleđuje *daemon* niti, te se program *while* petlje izvršava paralelno sa programom modula za balansiranje opterećenja.

Na sledećem dijagramu prikazan je LBORU algoritam *while* petlje.



Slika 11.2 LBORU algoritam *while* petlje

Овај Образац чини саставни део докторске дисертације, односно докторског уметничког пројекта који се брани на Универзитету у Новом Саду. Попуњен Образац укоричити иза текста докторске дисертације, односно докторског уметничког пројекта.

План третмана података

Назив пројекта/истраживања
Развој и имплементација LBORU методе за динамичко балансирање оптерећења сервера у хибридним SDN мрежама
Назив институције/институција у оквиру којих се спроводи истраживање
а) Факултет техничких наука, Универзитет у Новом Саду
Назив програма у оквиру ког се реализује истраживање
Истраживање се врши у оквиру израде докторске дисертације на студијском програму Енергетика, електроника и телекомуникације.
1. Опис података
1.1 Врста студије *Укратко описати тип студије у оквиру које се подаци прикупљају* Докторске академске студије у научној области електротехничког и рачунарског инжењерства (**Докторска дисертација**) 1.2 Врсте података а) **квантитативни** б) **квалитативни** 1.3. Начин прикупљања података а) анкете, упитници, тестови б) клиничке процене, медицински записи, електронски здравствени записи в) генотипови: навести врсту _____

г) административни подаци: навести врсту _____

д) узорци ткива: навести врсту_____

ђ) снимци, фотографије: навести врсту_____

е) текст, навести врсту _____

ж) мапа, навести врсту _____

з) остало: описати **рачунарски експерименти**

Развијен је алат за тестирање који шаље захтеве намењене *web* серверима и састоји се из две компоненте:

- компонента посла – успоставља везу са *web* сервером, извршава трансакцију једног захтева и извештава о времену потребном за њен завршетак. Процедура ове компоненте се извршава у бесконачној петљи како би се симулирао један конкурентни корисник, који стално обавља трансакцију.
- компонента конкуренције – повећава број истовремених корисника у контролисаним условима, записује време одзива сваке трансакције, и обавља статистичке рачунске операције.

Док компонента посла шаље захтеве *web* серверима, компонента конкуренције прикупља параметре тестирања као што су:

- број конкурентних веза
- просечно време потребно за обављање једног захтева
- број завршених трансакција у секунди

Снимање перформанси постављеног окружења за тестирање решења изложеног у поглављу 9, обављено је помоћу следећих алата:

- bwm-ng алат—користи се за праћење и евидентирање активне пропусности физичких интерфејса на страни *CarClient*-а (за потребе праћења *download*-а) і *DistributionNetwork* свича (за потребе праћења *upload*-а);
- D-ITG алат —користи се за евидентирање и анализу QoS параметара (кашњење, џитер и губитак пакета) UDP стримовања на страни пријемника на *CarClient*-у и SP3 serveru;
- tcpdump алат—користи се за прикупљање комплетних евиденција саобраћаја у процесу тестирања на *DistributionNetwork* свичу.

1.3 Формат података, употребљене скале, количина података

Рачунарски експерименти

1.3.1 Употребљени софтвер и формат датотеке:

У сврху прикупљања података примењен је претходно описан софтверки алат за тестирање (1.3. Начин прикупљања података).

a) Excel фајл, датотека **.xlsx, .ods**

b) SPSS фајл, датотека _____

c) PDF фајл, датотека _____

d) Текст фајл, датотека _____

e) JPG фајл, датотека _____

f) Остало, датотека **.png**

1.3.2. Број записа (код квантитативних података)

а) број варијабли **велики број**

б) број мерења (испитаника, процена, снимака и сл.) **велики број**

1.3.3. Поновљена мерења

а) да

б) **не**

Уколико је одговор да, одговорити на следећа питања:

а) временски размак између поновљених мера је: _____

б) варијабле које се више пута мере односе се на: _____

в) нове верзије фајлова који садрже поновљена мерења су именоване као _____

Напомене: _____

Да ли формати и софтвер омогућавају дељење и дугорочну валидност података?

а) ***Да***

б) Не

Ако је одговор не, образложити _____

_____

2. Прикупљање података

2.1 Методологија за прикупљање/генерисање података

2.1.1. У оквиру ког истраживачког нацрта су подаци прикупљени?

а) експеримент, навести тип **рачунарски екцперимент**

б) корелационо истраживање, навести тип _____

ц) анализа текста, навести тип **анализа доступне литературе**

д) остало, навести шта _____

2.1.2 Навести врсте мерних инструмената или стандарде података специфичних за одређену научну дисциплину (ако постоје).

_____

_____

2.2 Квалитет података и стандарди

2.2.1. Третман недостајућих података

а) Да ли матрица садржи недостајуће податке? Да **Не**

Ако је одговор да, одговорити на следећа питања:

а) Колики је број недостајућих података? _____

б) Да ли се кориснику матрице препоручује замена недостајућих података? Да Не

в) Ако је одговор да, навести сугестије за третман замене недостајућих података

_____

2.2.2. На који начин је контролисан квалитет података? Описати

Квалитет података је контролисан поређењем експерименталних и теоријских података.

2.2.3. На који начин је извршена контрола уноса података у матрицу?

Контрола уноса података је изведена на бази експертног знања.

3. Третман података и пратећа документација

3.1. Третман и чување података

3.1.1. Подаци ће бити депоновани у **Универзитет у Новом Саду** *репозиторијум.*

3.1.2. URL адреса https://www.cris.uns.ac.rs/searchDissertations.jsf

3.1.3. DOI _____

3.1.4. Да ли ће подаци бити у отвореном приступу?

а) ***Да***

б) *Да, али после ембарга који ће трајати до* _____

в) *Не*

Ако је одговор не, навести разлог _____

3.1.5. Подаци неће бити депоновани у репозиторијум, али ће бити чувани.

Образложење

_____

_____

3.2 Метаподаци и документација података

3.2.1. Који стандард за метаподатке ће бити примењен? **Стандард који примењује Репозиторијум докторских дисертација Универзитета у Новом Саду**

3.2.1. Навести метаподатке на основу којих су подаци депоновани у репозиторијум.

_____

_____

Ако је потребно, навести методе које се користе за преузимање података, аналитичке и процедуралне информације, њихово кодирање, детаљне описе варијабли, записа итд.

_____

_____

_____

_____

3.3 Стратегија и стандарди за чување података

3.3.1. До ког периода ће подаци бити чувани у репозиторијуму? _____

3.3.2. Да ли ће подаци бити депоновани под шифром? **Да** Не

3.3.3. Да ли ће шифра бити доступна одређеном кругу истраживача? **Да** Не

3.3.4. Да ли се подаци морају уклонити из отвореног приступа после извесног времена?

Да **Не**

Образложити

_____

_____

4. Безбедност података и заштита поверљивих информација

Овај одељак МОРА бити попуњен ако ваши подаци укључују личне податке који се односе на учеснике у истраживању. За друга истраживања треба такође размотрити заштиту и сигурност података.

4.1 Формални стандарди за сигурност информација/података

Истраживачи који спроводе испитивања с људима морају да се придржавају Закона о заштити података о личности *(https://www.paragraf.rs/propisi/zakon_o_zastiti_podataka_o_licnosti.html)* и одговарајућег институционалног кодекса о академском интегритету.

4.1.2. Да ли је истраживање одобрено од стране етичке комисије? Да **Не**

Ако је одговор Да, навести датум и назив етичке комисије која је одобрила истраживање

_____

4.1.2. Да ли подаци укључују личне податке учесника у истраживању? Да **Не**

Ако је одговор да, наведите на који начин сте осигурали поверљивост и сигурност информација везаних за испитанике:

а) Подаци нису у отвореном приступу

б) Подаци су анонимизирани

ц) Остало, навести шта

_____

5. Доступност података

5.1. Подаци ће бити

а) ***јавно доступни***

б) доступни само уском кругу истраживача у одређеној научној области

ц) затворени

Ако су подаци доступни само уском кругу истраживача, навести под којим условима могу да их користе:

_____

_____

Ако су подаци доступни само уском кругу истраживача, навести на који начин могу приступити подацима:

_____

_____

5.4. Навести лиценцу под којом ће прикупљени подаци бити архивирани.

ауторство – некомерцијално

6. Улоге и одговорност

6.1. Навести име и презиме и мејл адресу власника (аутора) података

Теодор Малбашић, malbasicteodor@gmail.com

6.2. Навести име и презиме и мејл адресу особе која одржава матрицу с подацима

Теодор Малбашић, malbasicteodor@gmail.com

6.3. Навести име и презиме и мејл адресу особе која омогућује приступ подацима другим истраживачима

Теодор Малбашић, malbasicteodor@gmail.com